NAME

linuxcnc − LinuxCNC (The Enhanced Machine Controller)

SYNOPSIS

linuxcnc [*-v*] [*-d*] [*INIFILE*]

DESCRIPTION

linuxcnc is used to start LinuxCNC (The Enhanced Machine Controller). It starts the realtime system and then initializes a number of LinuxCNC components (IO, Motion, GUI, HAL, etc). The most important parameter is *INIFILE*, which specifies the configuration name you would like to run. If *INIFILE* is not specified, the **linuxcnc** script presents a graphical wizard to let you choose one.

OPTIONS

−v Be a little bit verbose. This causes the script to print information as it works.

−d Print lots of debug information. All executed commands are echoed to the screen. This mode is useful when something is not working as it should.

INIFILE

The ini file is the main piece of an LinuxCNC configuration. It is not the entire configuration; there are various other files that go with it (NML files, HAL files, TBL files, VAR files). It is, however, the most important one, because it is the file that holds the configuration together. It can adjust a lot of parameters itself, but it also tells **linuxcnc** which other files to load and use.

There are several ways to specify which config to use:

Specify the absolute path to an ini, e.g.
linuxcnc */usr/local/linuxcnc/configs/sim/sim.ini*

Specify a relative path from the current directory, e.g.
linuxcnc *configs/sim/sim.ini*

Otherwise, in the case where the **INIFILE** is not specified, the behavior will depend on whether you configured linuxcnc with **--enable-run-in-place**. If so, the linuxcnc config chooser will search only the configs directory in your source tree. If not (or if you are using a packaged version of linuxcnc), it may search several directories. The config chooser is currently set to search the path:

~/linuxcnc/configs:/home/buildslave/emc2-buildbot/wheezy-amd64-clang/docs/build/configs

EXAMPLES

linuxcnc

linuxcnc *configs/sim/sim.ini*

linuxcnc */etc/linuxcnc/sample-configs/stepper/stepper_mm.ini*

SEE ALSO

halcmd(1)

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at /usr/share/doc/linuxcnc/.

HISTORY

BUGS

None known at this time.

AUTHOR

This man page written by Alex Joni, as part of the LinuxCNC Enhanced Machine Controller project.

REPORTING BUGS

Report bugs to alex_joni AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 Alex Joni.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

axis-remote − AXIS Remote Interface

SYNOPSIS

axis-remote *OPTIONS|FILENAME*

DESCRIPTION

axis-remote is a small script that triggers commands in a running AXIS GUI. Use **axis-remote --help** for further information.

OPTIONS

--ping, **-p**
: Check whether AXIS is running.

--reload, **-r**
: Make AXIS reload the currently loaded file.

--clear, **-c**
: Make AXIS clear the backplot.

--quit, **-q**
: Make AXIS quit.

--help, **-h**, **-?**
: Display a list of valid parameters for **axis-remote**.

--mdi COMMAND, **-m COMMAND**
: Run the MDI command **COMMAND**.

FILENAME
: Load the G-code file **FILENAME**.

SEE ALSO

axis(1)

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at /usr/share/doc/linuxcnc/.

HISTORY

BUGS

None known at this time.

AUTHOR

This man page written by Alex Joni, as part of the LinuxCNC project.

REPORTING BUGS

Report bugs to alex_joni AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2007 Alex Joni.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

axis − AXIS LinuxCNC Graphical User Interface

SYNOPSIS

axis -*ini INIFILE*

DESCRIPTION

axis is one of the Graphical User Interfaces (GUI) for LinuxCNC It gets run by the runscript usually.

OPTIONS

INIFILE

The ini file is the main piece of an LinuxCNC configuration. It is not the entire configuration; there are various other files that go with it (NML files, HAL files, TBL files, VAR files). It is, however, the most important one, because it is the file that holds the configuration together. It can adjust a lot of parameters itself, but it also tells **LinuxCNC** which other files to load and use.

SEE ALSO

LinuxCNC(1)

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at /usr/share/doc/LinuxCNC/.

HISTORY

BUGS

None known at this time.

AUTHOR

This man page written by Alex Joni, as part of the LinuxCNC project.

REPORTING BUGS

Report bugs to alex_joni AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2007 Alex Joni.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

comp − Build, compile and install LinuxCNC HAL components

SYNOPSIS

comp [**--compile**|**--preprocess**|**--document**|**--view-doc**] compfile...
sudo **comp** [**--install**|**--install-doc**] compfile...
comp --compile --userspace cfile...
sudo **comp --install --userspace** cfile...
sudo **comp --install --userspace** pyfile...

DESCRIPTION

comp performs many different functions:

- Compile **.comp** and **.c** files into **.so** or **.ko** HAL realtime components (the **--compile** flag)
- Compile **.comp** and **.c** files into HAL userspace components (the **--compile --userspace** flag)
- Preprocess **.comp** files into **.c** files (the **--preprocess** flag)
- Extract documentation from **.comp** files into **.9** manpage files (the **--document** flag)
- Display documentation from **.comp** files onscreen (the **--view-doc** flag)
- Compile and install **.comp** and **.c** files into the proper directory for HAL realtime components (the **--install** flag), which may require *sudo* to write to system directories.
- Install **.c** and **.py** files into the proper directory for HAL userspace components (the **--install --userspace** flag), which may require *sudo* to write to system directories.
- Extract documentation from **.comp** files into **.9** manpage files in the proper system directory (the **--install** flag), which may require *sudo* to write to system directories.
- Preprocess **.comp** files into **.c** files (the **--preprocess** flag)

SEE ALSO

Comp HAL Component Generator in the LinuxCNC documentation for a full description of the **.comp** syntax, along with examples

pydoc hal and *Creating Userspace Python Components* in the LinuxCNC documentation for documentation on the Python interface to HAL components

comp(9) for documentation on the "two input comparator with hysteresis", a HAL realtime component with the same name as this program

NAME

gladevcp − Virtual Control Panel for LinuxCNC based on Glade, Gtk and HAL widgets

SYNOPSIS

gladevcp [-g *WxH+X+Y*] [-c *component-name*] [-u *handler*] [-U *useroption*] [-H *halfile*] [-d] *myfile.ui*

OPTIONS

-g *WxH+X+Y*

This sets the initial geometry of the root window. Use 'WxH' for just size, '+X+Y' for just position, or 'WxH+X+Y' for both. Size / position use pixel units. Position is referenced from top left.

-c *component-name*

Use *component-name* as the HAL component name. If the component name is not specified, the basename of the ui file is used.

-u *handler*

Instructs gladevcp to inspect the Python script *handler* for event handlers, and connect them to signals in the ui file.

-U *useroption*

gladevcp collects all *useroption* strings and passes them to the handler init() method as a list of strings without further inspection.

-x *XID* Reparent gladevcp into an existing window *XID* instead of creating a new top level window.

-H *halfile*

gladevcp runs *halfile* - a list of HAL commands - by executing *halcmd -c halfile* after the HAL component is finalized.

-d enable debug output.

-R *gtkrcfile*

explicitly load a gtkrc file.

-t *THEME*

set gtk theme. Default is *system* theme. Different panels can have different themes.

-m *MAXIMUM*

force panel window to maxumize. Together with the *-g geometry* option one can move the panel to a second monitor and force it to use all of the screen

-R explicitly deactivate workaround for a gtk bug which makes matches of widget and widget_class matches in gtk theme and gtkrc files fail. Normally not needed.

SEE ALSO

GladeVCP in the LinuxCNC documentation for a description of gladevcp's capabilities and the associated HAL widget set, along with examples

NAME

gs2_vfd - HAL userspace component for Automation Direct GS2 VFD's

SYNOPSIS

gs2_vfd [OPTIONS]

DESCRIPTION

This manual page explains the **gs2_vfd** component. This component reads and writes to the GS2 via a modbus connection.

gs2_vfd is for use with LinuxCNC

OPTIONS

-b, --bits <n>
(default 8) Set number of data bits to <n>, where n must be from 5 to 8 inclusive

-d, --device <path>
(default /dev/ttyS0) Set the name of the serial device node to use.

-g, --debug
Turn on debugging messages. This will also set the verbose flag. Debug mode will cause all modbus messages to be printed in hex on the terminal.

-n, --name <string>
(default gs2_vfd) Set the name of the HAL module. The HAL comp name will be set to <string>, and all pin and parameter names will begin with <string>.

-p, --parity [even,odd,none]
(default odd) Set serial parity to even, odd, or none.

-r, --rate <n>
(default 38400) Set baud rate to <n>. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

-s, --stopbits [1,2]
(default 1) Set serial stop bits to 1 or 2

-t, --target <n>
(default 1) Set MODBUS target (slave) number. This must match the device number you set on the GS2.

-v, --verbose
Turn on debug messages. Note that if there are serial errors, this may become annoying. At the moment it doesn't make much difference most of the time.

PINS

<name>.DC-bus-volts (float, out)
from the VFD

<name>.at-speed (bit, out)
when drive is at commanded speed

<name>.err-reset (bit, in)
reset errors sent to VFD

<name>.firmware-revision (s32, out)
from the VFD

<name>.frequency-command (float, out)
from the VFD

<name>.frequency-out (float, out)
from the VFD

<name>.is-stopped (bit, out)
when the VFD reports 0 Hz output

<name>.load-percentage (float, out)
from the VFD

<name>.motor-RPM (float, out)
from the VFD

<name>.output-current (float, out)
from the VFD

<name>.output-voltage (float, out)
from the VFD

<name>.power-factor (float, out)
from the VFD

<name>.scale-frequency (float, out)
from the VFD

<name>.speed-command (float, in)
speed sent to VFD in RPM It is an error to send a speed faster than the Motor Max RPM as set in the VFD

<name>.spindle-fwd (bit, in)
1 for FWD and 0 for REV sent to VFD

<name>.spindle-on (bit, in)
1 for ON and 0 for OFF sent to VFD, only on when running

<name>.spindle-rev (bit, in)
1 for ON and 0 for OFF, only on when running

<name>.status-1 (s32, out)
Drive Status of the VFD (see the GS2 manual)

<name>.status-2 (s32, out)
Drive Status of the VFD (see the GS2 manual) Note that the value is a sum of all the bits that are on. So a 163 which means the drive is in the run mode is the sum of 3 (run) + 32 (freq set by serial) + 128 (operation set by serial).

PARAMETERS

<name>.error-count (s32, RW)

<name>.loop-time (float, RW)
how often the modbus is polled (default 0.1)

<name>.nameplate-HZ (float, RW)
Nameplate Hz of motor (default 60)

<name>.nameplate-RPM (float, RW)
Nameplate RPM of motor (default 1730)

<name>.retval (s32, RW)
the return value of an error in HAL

<name>.tolerance (float, RW)
speed tolerance (default 0.01)

<name>.ack-delay (s32, RW)
number of read/write cycles before checking at-speed (default 2)

SEE ALSO

GS2 Driver in the LinuxCNC documentation for a full description of the **GS2** syntax

GS2 Examples in the LinuxCNC documentation for examples using the **GS2** component

BUGS

AUTHOR

John Thornton

LICENSE

GPL

NAME

hal_input − control HAL pins with any Linux input device, including USB HID devices

SYNOPSIS

loadusr **hal_input** *[-KRAL] inputspec ...*

DESCRIPTION

hal_input is an interface between HAL and any Linux input device, including USB HID devices. For each device named, **hal_input** creates pins corresponding to its keys, absolute axes, and LEDs. At a fixed rate of approximately 10ms, it synchronizes the device and the HAL pins.

INPUT SPECIFICATION

The *inputspec* may be in one of several forms:

A string *S*

A substring or shell-style pattern match will be tested against the "name" of the device, the "phys" (which gives information about how it is connected), and the "id", which is a string of the form "Bus=... Vendor=... Product=... Version=...". You can view the name, phys, and id of attached devices by executing **less /proc/bus/input/devices**. Examples:

SpaceBall
"Vendor=001f Product=0001"
serio*/input0

A number *N*

This opens **/dev/input/event***N*. Except for devices that are always attached to the system, this number may change over reboots or when the device is removed. For this reason, using an integer is not recommended.

When several devices are identified by the same string, add "**:***N*" where *N* is the index of the desired device. For example, if **Mouse** matches **input3** and **input10**, then **Mouse** and **Mouse:0** select **input3**. Specifying **mouse:1** selects input10.

For devices that appear as multiple entries in /dev/input, these indices are likely to stay the same every time. For multiple identical devices, these indices are likely to depend on the insertion order, but stay the same across reboots as long as the devices are not moved to different ports or unplugged while the machine is booted.

If the first character of the *inputspec* is a "+", then **hal_input** requests exclusive access to the device. The first device matching an *inputspec* is used. Any number of *inputspec*s may be used.

A *subset option may preceed each inputspec*. The subset option begins with a dash. Each letter in the subset option specifies a device feature to **include**. Features that are not specified are excluded. For instance, to export keyboard LEDs to HAL without exporting keys, use

hal_input -L *keyboard ...*

DEVICE FEATURES SUPPORTED

- EV_KEY (buttons and keys). Subset -K
- EV_ABS (absolute analog inputs). Subset -A
- EV_REL (relative analog inputs). Subset -R
- EV_LED (LED outputs). Subset -L

HAL PINS AND PARAMETERS

For buttons

input.*N***.btn-***name* bit out
input.*N***.btn-***name***-not** bit out

Created for each button on the device.

For keys

input.*N*.key-*name*
input.*N*.key-*name*-not
Created for each key on the device.

For absolute axes

input.*N*.abs-*name*-counts s32 out
input.*N*.abs-*name*-position float out
input.*N*.abs-*name*-scale parameter float rw
input.*N*.abs-*name*-offset parameter float rw
input.*N*.abs-*name*-fuzz parameter s32 rw
input.*N*.abs-*name*-flat parameter s32 rw
input.*N*.abs-*name*-min parameter s32 r
input.*N*.abs-*name*-max parameter s32 r
Created for each absolute axis on the device. Device positions closer than **flat** to **offset** are reported as **offset** in **counts**, and **counts** does not change until the device position changes by at least **fuzz**. The position is computed as **position** = (**counts** - **offset**) / **scale**. The default value of **scale** and **offset** map the range of the axis reported by the operating system to [-1,1]. The default values of **fuzz** and **flat** are those reported by the operating system. The values of **min** and **max** are those reported by the operating system.

For relative axes

input.*N*.rel-*name*-counts s32 out
input.*N*.rel-*name*-position float out
input.*N*.rel-*name*-reset bit in
input.*N*.rel-*name*-scale parameter float rw
input.*N*.rel-*name*-absolute parameter s32 rw
input.*N*.rel-*name*-precision parameter s32 rw
input.*N*.rel-*name*-last parameter s32 rw
Created for each relative axis on the device. As long as **reset** is true, **counts** is reset to zero regardless of any past or current axis movement. Otherwise, **counts** increases or decreases according to the motion of the axis. **counts** is divided by position-scale to give **position**. The default value of **position** is 1. There are some devices, notably scroll wheels, which return signed values with less resolution than 32 bits. The default value of **precision** is 32. **precision** can be set to 8 for a device that returns signed 8 bit values, or any other value from 1 to 32. **absolute**, when set true, ignores duplicate events with the same value. This allows for devices that repeat events without any user action to work correctly. **last** shows the most recent count value returned by the device, and is used in the implementation of **absolute**.

For LEDs

input.*N*.led-*name* bit out
input.*N*.led-*name*-invert parameter bit rw
Created for each LED on the device.

PERMISSIONS AND UDEV

By default, the input devices may not be accessible to regular users--**hal_input** requires read-write access, even if the device has no outputs. To change the default permission of a device, add a new file to /etc/udev/rules.d to set the device's GROUP to "plugdev". You can do this for all input devices with this rule:

SUBSYSTEM=="input", MODE="0660", GROUP="plugdev"

You can also make more specific rules for particular devices. For instance, a SpaceBall input device uses the 'spaceball' kernel module, so a udev entry for it would read:

DRIVER=="spaceball", MODE="0660", GROUP="plugdev"

the next time the device is attached to the system, it will be accessible to the "plugdev" group.

For USB devices, the udev line would refer to the device's Vendor and Product values, such as

SYSFS{idProduct}=="c00e", SYSFS{idVendor}=="046d", MODE="0660", GROUP="plugdev"

for a particular logictech-brand mouse.

For more information on writing udev rules, see **udev(8)**.

BUGS

The initial state of keys, buttons, and absolute axes are erroneously reported as FALSE or 0 until an event is received for that key, button, or axis.

SEE ALSO

udev(8)

NAME

halcmd − manipulate the LinuxCNC HAL from the command line

SYNOPSIS

halcmd [*OPTIONS*] [*COMMAND* [*ARG*]]

halrun [*-I*] [*HALCMD OPTIONS*]

halrun [*-U*]

DESCRIPTION

halcmd is used to manipulate the HAL (Hardware Abstraction Layer) from the command line. **halcmd** can optionally read commands from a file, allowing complex HAL configurations to be set up with a single command.

halrun is a convenience script which sets up the realtime environment, executes **halcmd** with the given arguments, optionally runs an interactive **halcmd -kf** if *-I* is given, then tears down the realtime environment.

If the **readline** library is available when LinuxCNC is compiled, then **halcmd** offers commandline editing and completion when running interactively. Use the up arrow to recall previous commands, and press tab to complete the names of items such as pins and signals.

OPTIONS

-I Before tearing down the realtime environment, run an interactive halcmd. **halrun** only. If **-I** is used, it must precede all other commandline arguments.

−f [*file*] Ignore commands on command line, take input from *file* instead. If *file* is not specified, take input from *stdin*.

-i *inifile*
Use variables from *inifile* for substitutions. See **SUBSTITUTION** below.

−k Keep going after failed command(s). The default is to stop and return failure if any command fails.

−q display errors only (default)

−Q display nothing, execute commands silently

−s Script-friendly mode. In this mode, *show* will not output titles for the items shown. Also, module names will be printed instead of ID codes in pin, param, and funct listings. Threads are printed on a single line, with the thread period, FP usage and name first, followed by all of the functions in the thread, in execution order. Signals are printed on a single line, with the type, value, and signal name first, followed by a list of pins connected to the signal, showing both the direction and the pin name. No prompt will be printed if both **-s** and **-f** are specified.

-R Release the HAL mutex. This is useful for recovering when a HAL component has crashed while holding the HAL mutex.

-U Forcibly cause the realtime environment to exit. It releases the HAL mutex, requests that all HAL components unload, and stops the realtime system. **halrun** only. **-U** must be the only command-line argument.

−v display results of each command

−V display lots of debugging junk

−h [*command*]
display a help screen and exit, displays extended help on *command* if specified

COMMANDS

Commands tell **halcmd** what to do. Normally **halcmd** reads a single command from the command line and executes it. If the '**-f**' option is used to read commands from a file, **halcmd** reads each line of the file as a new command. Anything following '**#**' on a line is a comment.

loadrt *modname*

(*load r*eal*t*ime module) Loads a realtime HAL module called *modname*. **halcmd** looks for the module in a directory specified at compile time.

In systems with realtime, **halcmd** calls the **linuxcnc_module_helper** to load realtime modules. **linuxcnc_module_helper** is a setuid program and is compiled with a whitelist of modules it is allowed to load. This is currently just a list of **LinuxCNC**-related modules. The **linuxcnc_module_helper** execs insmod, so return codes and error messages are those from insmod. Administrators who wish to restrict which users can load these **LinuxCNC**-related kernel modules can do this by setting the permissions and group on **linuxcnc_module_helper** appropriately.

In systems without realtime **halcmd** calls the **rtapi_app** which creates the simulated realtime environment if it did not yet exist, and then loads the requested component with a call to **dlopen(3)**.

unloadrt *modname*

(*unload r*eal*t*ime module) Unloads a realtime HAL module called *modname*. If *modname* is "all", it will unload all currently loaded realtime HAL modules. **unloadrt** also works by execing **linuxcnc_module_helper** or **rtapi_app**, just like **loadrt**.

loadusr *[flags] unix-command*

(*load Us*er*s*pace component) Executes the given *unix-command*, usually to load a userspace component. *[flags]* may be one or more of:

- **-W** to wait for the component to become ready. The component is assumed to have the same name as the first argument of the command.
- **-Wn name** to wait for the component, which will have the given name.
- **-w** to wait for the program to exit
- **-i** to ignore the program return value (with -w)

waitusr *name*

(*wait* for *Us*er*s*pace component) Waits for user space component *name* to disconnect from HAL (usually on exit). The component must already be loaded. Usefull near the end of a HAL file to wait until the user closes some user interface component before cleaning up and exiting.

unloadusr *compname*

(*unload Us*er*s*pace component) Unloads a userspace component called *compname*. If *compname* is "all", it will unload all userspace components. **unloadusr** works by sending SIGTERM to all userspace components.

unload *compname*

Unloads a userspace component or realtime module. If *compname* is "all", it will unload all userspace components and realtime modules.

newsig *signame type*

(*new sig*nal) Creates a new HAL signal called *signame* that may later be used to connect two or more HAL component pins. *type* is the data type of the new signal, and must be one of "**bit**", "**s32**", "**u32**", or "**float**". Fails if a signal of the same name already exists.

delsig *signame*

(*del*ete *sig*nal) Deletes HAL signal *signame*. Any pins currently linked to the signal will be unlinked. Fails if *signame* does not exist.

sets *signame value*

(*set* signal) Sets the value of signal *signame* to *value*. Fails if *signame* does not exist, if it already has a writer, or if *value* is not a legal value. Legal values depend on the signals's type.

stype *name*

(signal type) Gets the type of signal *name*. Fails if *name* does not exist as a signal.

gets *signame*

(*get* signal) Gets the value of signal *signame*. Fails if *signame* does not exist.

linkps *pinname* [*arrow*] *signame*

(*link p*in to signal) Establishs a link between a HAL component pin *pinname* and a HAL signal *signame*. Any previous link to *pinname* will be broken. *arrow* can be "=>", "<=", "<=>", or omitted. **halcmd** ignores arrows, but they can be useful in command files to document the direction of data flow. Arrows should not be used on the command line since the shell might try to interpret them. Fails if either *pinname* or *signame* does not exist, or if they are not the same type type.

linksp *signame* [*arrow*] *pinname*

(*link* signal to *p*in) Works like **linkps** but reverses the order of the arguments. **halcmd** treats both link commands exactly the same. Use whichever you prefer.

linkpp *pinname1* [*arrow*] *pinname2*

(OBSOLETE - use **net** instead) (*link p*in to *p*in) Shortcut for **linkps** that creates the signal (named like the first pin), then links them both to that signal. **halcmd** treats this just as if it were:

halcmd newsig pinname1
halcmd linksp pinname1 pinname1
halcmd linksp pinname1 pinname2

net *signame pinname ...*

Create *signname* to match the type of *pinname* if it does not yet exist. Then, link *signame* to each *pinname* in turn. Arrows may be used as in **linkps**.

unlinkp *pinname*

(*unlink p*in) Breaks any previous link to *pinname*. Fails if *pinname* does not exist.

setp *name value*

(*set p*arameter or *p*in) Sets the value of parameter or pin *name* to *value*. Fails if *name* does not exist as a pin or parameter, if it is a parameter that is not writable, if it is a pin that is an output, if it is a pin that is already attached to a signal, or if *value* is not a legal value. Legal values depend on the type of the pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

paramname **=** *value*

pinname **=** *value*

Identical to **setp**. This alternate form of the command may be more convenient and readable when used in a file.

ptype *name*

(*p*arameter or *p*in *type*) Gets the type of parameter or pin *name*. Fails if *name* does not exist as a pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

getp *name*

(*get p*arameter or *p*in) Gets the value of parameter or pin *name*. Fails if *name* does not exist as a pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

addf *functname threadname*

(*add f*unction) Adds function *functname* to realtime thread *threadname*. *functname* will run after any functions that were previously added to the thread. Fails if either *functname* or *threadname* does not exist, or if they are incompatible.

delf *functname threadname*

(*del*ete *f*unction) Removes function *functname* from realtime thread *threadname*. Fails if either *functname* or *threadname* does not exist, or if *functname* is not currently part of *threadname*.

start Starts execution of realtime threads. Each thread periodically calls all of the functions that were added to it with the **addf** command, in the order in which they were added.

stop Stops execution of realtime threads. The threads will no longer call their functions.

show [*item*]
Prints HAL items to *stdout* in human readable format. *item* can be one of "**comp**" (components), "**pin**", "**sig**" (signals), "**param**" (parameters), "**funct**" (functions), or "**thread**". The type "**all**" can be used to show matching items of all the preceeding types. If *item* is omitted, **show** will print everything.

item This is equivalent to **show all [item]**.

save [*item*]
Prints HAL items to *stdout* in the form of HAL commands. These commands can be redirected to a file and later executed using **halcmd -f** to restore the saved configuration. *item* can be one of the following: "**comp**" generates a **loadrt** command for realtime component. "**sig**" generates a **newsig** command for each signal, and "**sigu**" generates a **newsig** command for each unlinked signal (for use with **netl** and **netla**). "**link**" and "**linka**" both generate **linkps** commands for each link. (**linka** includes arrows, while **link** does not.) "**net**" and "**neta**" both generate one **newsig** command for each signal, followed by **linksp** commands for each pin linked to that signal. (**neta** includes arrows.) "**netl**" generates one **net** command for each linked signal, and "**netla**" generates a similar command using arrows. "**param**" generates one **setp** command for each parameter. "**thread**" generates one **addf** command for each function in each realtime thread. If *item* is omitted, **save** does the equivalent of **comp**, **sigu**, **link**, **param**, and **thread**.

source *filename.hal*
Execute the commands from *filename.hal*.

SUBSTITUTION

After a command is read but before it is executed, several types of variable substitution take place.

Environment Variables

Environment variables have the following formats:

$ENVVAR followed by end-of-line or whitespace

$(ENVVAR)

Inifile Variables

Inifile variables are available only when an inifile was specified with the halcmd **-i** flag. They have the following formats:

[SECTION]VAR followed by end-of-line or whitespace

[SECTION](VAR)

EXAMPLES

SEE ALSO

HISTORY

BUGS

None known at this time.

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Now includes major contributions by several members of the project.

REPORTING BUGS

Report bugs to the LinuxCNC bug tracker ⟨`http://sf.net/tracker/?group_id=6744&atid=106744`⟩.

COPYRIGHT

Copyright © 2003 John Kasunich.
This is free software; see the source for copying conditions. There is NO warranty; not even for

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

halmeter − observe HAL pins, signals, and parameters

SYNOPSIS

halmeter [**-s**] [**pin**|**sig**|**param** *name*] [**-g** *X-positon Y-position [Width]*]

DESCRIPTION

halmeter is used to observe HAL (Hardware Abstraction Layer) pins, signals, or parameters. It serves the same purpose as a multimeter does when working on physical systems.

OPTIONS

pin *name*
display the HAL pin *name*.

sig *name*
display the HAL signal *name*.

param *name*
display the HAL parameter *name*.

If neither **pin**, **sig**, or **param** are specified, the
window starts out blank and the user must select an item to observe.

−s small window. Non-interactive, must be used with **pin**, **sig**, or **param** to select the item to display. The item name is displayed in the title bar instead of the window, and there are no "Select" or "Exit" buttons. Handy when you want a lot of meters in a small space.

−g geometry position. allows one to specify the intial starting position and optionally the width of the meter. Referenced from top left of screen in pixel units. Handy when you want to load a lot of meters in a script with out them displaying on top of each other.

USAGE

Unless **−s** is specified, there are two buttons, "Select" and "Exit". "Select" opens a dialog box to select the item (pin, signal, or parameter) to be observed. "Exit" does what you expect.

The selection dialog has "OK" "Apply", and "Cancel" buttons. OK displays the selected item and closes the dialog. "Apply" displays the selected item but keeps the selection dialog open. "Cancel" closes the dialog without changing the displayed item.

EXAMPLES

halmeter
Opens a meter window, with nothing initially displayed. Use the "Select" button to choose an item to observe. Does not return until the window is closed.

halmeter &
Open a meter window, with nothing initially displayed. Use the "Select" button to choose an item. Runs in the background leaving the shell free for other commands.

halmeter pin *parport.0.pin-03-out* **&**
Open a meter window, initially displaying HAL pin *parport.0.pin-03-out*. The "Select" button can be used to display other items. Runs in background.

halmeter -s pin *parport.0.pin-03-out* **&**
Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. The displayed item cannot be changed. Runs in background.

halmeter -s pin *parport.0.pin-03-out* **-g** *100 500* **&**
Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. places it 100 pixels to the left and 500 pixels down from top of screen. The displayed item cannot be changed. Runs in background.

halmeter -s pin *parport.0.pin-03-out* **-g** *100 500 400* **&**

Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. places it 100 pixels to the left and 500 pixels down from top of screen. The width will be 400 pixels (270 is default) The displayed item cannot be changed. Runs in background.

SEE ALSO

HISTORY

BUGS

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

REPORTING BUGS

Report bugs to jmkasunich AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2003 John Kasunich.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

halsampler − sample data from HAL in realtime

SYNOPSIS

halsampler [*options*]

DESCRIPTION

sampler(9) and **halsampler** are used together to sample HAL data in real time and store it in a file. **sampler** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. It then begins sampling data from the HAL and storing it to the FIFO. **halsampler** is a user space program that copies data from the FIFO to stdout, where it can be redirected to a file or piped to some other program.

OPTIONS

-c *CHAN*

instructs **halsampler** to read from FIFO *CHAN*. FIFOs are numbered from zero, and the default value is zero, so this option is not needed unless multiple FIFOs have been created.

-n *COUNT*

instructs **halsampler** to read *COUNT* samples from the FIFO, then exit. If **-n** is not specified, **halsampler** will read continuously until it is killed.

-t instructs **halsampler** to tag each line by printing the sample number in the first column.

FILENAME

instructs **halsampler** to write to **FILENAME** instead of to stdout.

USAGE

A FIFO must first be created by loading **sampler**(9) with **halcmd loadrt** or a **loadrt** command in a .hal file. Then **halsampler** can be invoked to begin printing data from the FIFO to stdout.

Data is printed one line per sample. If **-t** was specified, the sample number is printed first. The data follows, in the order that the pins were defined in the config string. For example, if the **sampler** config string was "ffbs" then a typical line of output (without **-t**) would look like:

123.55 33.4 0 -12

halsampler prints data as fast as possible until the FIFO is empty, then it retries at regular intervals, until it is either killed or has printed *COUNT* samples as requested by **-n**. Usually, but not always, data printed by **halsampler** will be redirected to a file or piped to some other program.

The FIFO size should be chosen to absorb samples captured during any momentary disruptions in the flow of data, such as disk seeks, terminal scrolling, or the processing limitations of subsequent program in a pipeline. If the FIFO gets full and **sampler** is forced to overwrite old data, **halsampler** will print 'overrun' on a line by itself to mark each gap in the sampled data. If **-t** was specified, gaps in the sequential sample numbers in the first column can be used to determine exactly how many samples were lost.

The data format for **halsampler** output is the same as for **halstreamer**(1) input, so 'waveforms' captured with **halsampler** can be replayed using **halstreamer**. The **-t** option should not be used in this case.

EXIT STATUS

If a problem is encountered during initialization, **halsampler** prints a message to stderr and returns failure.

Upon printing *COUNT* samples (if **-n** was specified) it will shut down and return success. If it is terminated before printing the specified number of samples, it returns failure. This means that when **-n** is not specified, it will always return failure when terminated.

SEE ALSO

sampler(9) **streamer**(9) **halstreamer**(1)

HISTORY

BUGS

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

REPORTING BUGS

Report bugs to jmkasunich AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 John Kasunich.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

halstreamer − stream file data into HAL in real time

SYNOPSIS

halstreamer [*options*]

DESCRIPTION

streamer(9) and **halstreamer** are used together to stream data from a file into the HAL in real time. **streamer** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. **hal_streamer** is a user space program that copies data from stdin into the FIFO, so that **streamer** can write it to the HAL pins.

OPTIONS

-c *CHAN*

instructs **halstreamer** to write to FIFO *CHAN*. FIFOs are numbered from zero, and the default value is zero, so this option is not needed unless multiple FIFOs have been created.

FILENAME

instructs **halsampler** to read from **FILENAME** instead of from stdin.

USAGE

A FIFO must first be created by loading **streamer**(9) with **halcmd loadrt** or a **loadrt** command in a .hal file. Then **halstreamer** can be invoked to begin writing data into the FIFO.

Data is read from stdin, and is almost always either redirected from a file or piped from some other program, since keyboard input would be unable to keep up with even slow streaming rates.

Each line of input must match the pins that are attached to the FIFO, for example, if the **streamer** config string was "ffbs" then each line of input must consist of two floats, a bit, and a signed integer, in that order and separated by whitespace. Floats must be formatted as required by **strtod**(3), signed and unsigned integers must be formated as required by **strtol**(3) and **strtoul**(3), and bits must be either '0' or '1'.

halstreamer transfers data to the FIFO as fast as possible until the FIFO is full, then it retries at regular intervals, until it is either killed or reads **EOF** from stdin. Data can be redirected from a file or piped from some other program.

The FIFO size should be chosen to ride through any momentary disruptions in the flow of data, such as disk seeks. If the FIFO is big enough, **halstreamer** can be restarted with the same or a new file before the FIFO empties, resulting in a continuous stream of data.

The data format for **halstreamer** input is the same as for **halsampler**(1) output, so 'waveforms' captured with **halsampler** can be replayed using **halstreamer.**

EXIT STATUS

If a problem is encountered during initialization, **halstreamer** prints a message to stderr and returns failure.

If a badly formatted line is encountered while writing to the FIFO, it prints a message to stderr, skips the line, and continues (this behavior may be revised in the future).

Upon reading **EOF** from the input, it returns success. If it is terminated before the input ends, it returns failure.

SEE ALSO

streamer(9) **sampler**(9) **halsampler**(1)

HISTORY

BUGS

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

REPORTING BUGS

Report bugs to jmkasunich AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 John Kasunich.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

halui − observe HAL pins and command LinuxCNC through NML

SYNOPSIS

halui [**-ini <path-to-ini>**]

DESCRIPTION

halui is used to build a User Interface using hardware knobs and switches. It exports a big number of pins, and acts accordingly when these change.

OPTIONS

-ini name

use the *name* as the configuration file. Note: halui must find the nml file specified in the ini, usually that file is in the same folder as the ini, so it makes sense to run halui from that folder.

USAGE

When run, **halui** will export a large number of pins. A user can connect those to his physical knobs & switches & leds, and when a change is noticed halui triggers an appropriate event.

halui expects the signals to be debounced, so if needed (bad knob contact) connect the physical button to a HAL debounce filter first.

PINS

abort

halui.abort bit in
pin for clearing most errors

tool

halui.tool.length-offset.a float out
current applied tool length offset for the A axis

halui.tool.length-offset.b float out
current applied tool length offset for the B axis

halui.tool.length-offset.c float out
current applied tool length offset for the C axis

halui.tool.length-offset.u float out
current applied tool length offset for the U axis

halui.tool.length-offset.v float out
current applied tool length offset for the V axis

halui.tool.length-offset.w float out
current applied tool length offset for the W axis

halui.tool.length-offset.x float out
current applied tool length offset for the X axis

halui.tool.length-offset.y float out
current applied tool length offset for the Y axis

halui.tool.length-offset.z float out
current applied tool length offset for the Z axis

halui.tool.number u32 out
current selected tool

spindle

halui.spindle.brake-is-on bit out
status pin that tells us if brake is on

halui.spindle.brake-off bit in
pin for deactivating the spindle brake

halui.spindle.brake-on bit in
pin for activating the spindle brake

halui.spindle.decrease bit in
a rising edge on this pin decreases the current spindle speed by 100

halui.spindle.forward bit in
a rising edge on this pin makes the spindle go forward

halui.spindle.increase bit in
a rising edge on this pin increases the current spindle speed by 100

halui.spindle.is-on bit out
status pin telling if the spindle is on

halui.spindle.reverse bit in
a rising edge on this pin makes the spindle go reverse

halui.spindle.runs-backward bit out
status pin telling if the spindle is running backward

halui.spindle.runs-forward bit out
status pin telling if the spindle is running forward

halui.spindle.start bit in
a rising edge on this pin starts the spindle

halui.spindle.stop bit in
a rising edge on this pin stops the spindle

spindle override

halui.spindle-override.count-enable bit in (default: **TRUE**)
When TRUE, modify spindle override when counts changes.

halui.spindle-override.counts s32 in
counts X scale = spindle override percentage

halui.spindle-override.decrease bit in
pin for decreasing the SO (-=scale)

halui.spindle-override.direct-value bit in
pin to enable direct spindle override value input

halui.spindle-override.increase bit in
pin for increasing the SO (+=scale)

halui.spindle-override.scale float in
pin for setting the scale of counts for SO

halui.spindle-override.value float out
current FO value

program

halui.program.block-delete.is-on bit out
status pin telling that block delete is on

halui.program.block-delete.off bit in
pin for requesting that block delete is off

halui.program.block-delete.on bit in
pin for requesting that block delete is on

halui.program.is-idle bit out
status pin telling that no program is running

halui.program.is-paused bit out
status pin telling that a program is paused

halui.program.is-running bit out
status pin telling that a program is running

halui.program.optional-stop.is-on bit out
status pin telling that the optional stop is on

halui.program.optional-stop.off bit in
pin requesting that the optional stop is off

halui.program.optional-stop.on bit in
pin requesting that the optional stop is on

halui.program.pause bit in
pin for pausing a program

halui.program.resume bit in
pin for resuming a program

halui.program.run bit in
pin for running a program

halui.program.step bit in
pin for stepping in a program

halui.program.stop bit in
pin for stopping a program (note: this pin does the same thing as halui.abort)

mode

halui.mode.auto bit in
pin for requesting auto mode

halui.mode.is-auto bit out
pin for auto mode is on

halui.mode.is-joint bit out
pin showing joint by joint jog mode is on

halui.mode.is-manual bit out
pin for manual mode is on

halui.mode.is-mdi bit out
pin for mdi mode is on

halui.mode.is-teleop bit out
pin showing coordinated jog mode is on

halui.mode.joint bit in
pin for requesting joint by joint jog mode

halui.mode.manual bit in
pin for requesting manual mode

halui.mode.mdi bit in
pin for requesting mdi mode

halui.mode.teleop bit in
pin for requesting coordinated jog mode

mdi (optional)

halui.mdi-command-XX bit in
halui looks for ini variables named [HALUI]MDI_COMMAND, and exports a pin for each command it finds. When the pin is driven TRUE, **halui** runs the specified MDI command. XX is a two digit number starting at 00. If no [HALUI]MDI_COMMAND variables are set in the ini file, no halui.mdi-command-XX pins will be exported by halui.

mist

halui.mist.is-on bit out
pin for mist is on

halui.mist.off bit in
pin for stopping mist

halui.mist.on bit in
pin for starting mist

max-velocity

halui.max-velocity.count-enable bit in (default: **TRUE**)
When TRUE, modify max velocity when counts changes.

halui.max-velocity.counts s32 in
counts from an encoder for example to change maximum velocity

halui.max-velocity.decrease bit in
pin for decreasing the maximum velocity (-=scale)

halui.max-velocity.direct-value bit in
pin for using a direct value for max velocity

halui.max-velocity.increase bit in
pin for increasing the maximum velocity (+=scale)

halui.max-velocity.scale float in
pin for setting the scale on changing the maximum velocity

halui.max-velocity.value float out
Current value for maximum velocity

machine

halui.machine.is-on bit out
pin for machine is On/Off

halui.machine.off bit in
pin for setting machine Off

halui.machine.on bit in
pin for setting machine On

lube

halui.lube.is-on bit out
pin for lube is on

halui.lube.off bit in
pin for stopping lube

halui.lube.on bit in
pin for starting lube

joint

halui.joint.N.has-fault bit out
status pin telling that joint N has a fault

halui.joint.N.home bit in
pin for homing joint N

halui.joint.N.is-homed bit out
status pin telling that joint N is homed

halui.joint.N.is-selected bit out
status pin that joint N is selected

halui.joint.N.on-hard-max-limit bit out
status pin telling that joint N is on the positive hardware limit

halui.joint.N.on-hard-min-limit bit out
status pin telling that joint N is on the negative hardware limit

halui.joint.N.on-soft-max-limit bit out
status pin telling that joint N is on the positive software limit

halui.joint.N.on-soft-min-limit bit out
status pin telling that joint N is on the negative software limit

halui.joint.N.select bit in
pin for selecting joint N

halui.joint.N.unhome bit in
pin for unhoming joint N

halui.joint.selected u32 out
selected joint

halui.joint.selected.has-fault bit out
status pin selected joint is faulted

halui.joint.selected.home bit in
pin for homing the selected joint

halui.joint.selected.is-homed bit out
status pin telling that the selected joint is homed

halui.joint.selected.on-hard-max-limit bit out
status pin telling that the selected joint is on the positive hardware limit

halui.joint.selected.on-hard-min-limit bit out
status pin telling that the selected joint is on the negative hardware limit

halui.joint.selected.on-soft-max-limit bit out
status pin telling that the selected joint is on the positive software limit

halui.joint.selected.on-soft-min-limit bit out
status pin telling that the selected joint is on the negative software limit

halui.joint.selected.unhome bit in
pin for unhoming the selected joint

jog

halui.jog.deadband float in
pin for setting jog analog deadband (jog analog inputs smaller/slower than this are ignored)

halui.jog-speed float in
pin for setting jog speed for plus/minus jogging.

halui.jog.N.analog float in
pin for jogging the axis N using an float value (e.g. joystick)

halui.jog.N.increment float in
pin for setting the jog increment for axis N when using increment-plus/minus

halui.jog.N.increment-minus bit in
a rising edge will will make axis N jog in the negative direction by the increment amount

halui.jog.N.increment-plus bit in
a rising edge will will make axis N jog in the positive direction by the increment amount

halui.jog.N.minus bit in
pin for jogging axis N in negative direction at the halui.jog-speed velocity

halui.jog.N.plus bit in
pin for jogging axis N in positive direction at the halui.jog-speed velocity

halui.jog.selected.increment float in
pin for setting the jog increment for the selected axis when using increment-plus/minus

halui.jog.selected.increment-minus bit in
a rising edge will will make the selected axis jog in the negative direction by the increment amount

halui.jog.selected.increment-plus bit in
a rising edge will will make the selected axis jog in the positive direction by the increment amount

halui.jog.selected.minus bit in
pin for jogging the selected axis in negative direction at the halui.jog-speed velocity

halui.jog.selected.plus
pin for jogging the selected axis bit in in positive direction at the halui.jog-speed velocity

flood

halui.flood.is-on bit out
pin for flood is on

halui.flood.off bit in
pin for stopping flood

halui.flood.on bit in
pin for starting flood

feed override

halui.feed-override.count-enable bit in (default: **TRUE**)
When TRUE, modify feed override when counts changes.

halui.feed-override.counts s32 in
counts X scale = feed override percentage

halui.feed-override.decrease bit in
pin for decreasing the FO (-=scale)

halui.feed-override.direct-value bit in
pin to enable direct value feed override input

halui.feed-override.increase bit in
pin for increasing the FO (+=scale)

halui.feed-override.scale float in
pin for setting the scale on changing the FO

halui.feed-override.value float out
current Feed Override value

estop

halui.estop.activate bit in
pin for setting Estop (LinuxCNC internal) On

halui.estop.is-activated bit out
pin for displaying Estop state (LinuxCNC internal) On/Off

halui.estop.reset bit in
pin for resetting Estop (LinuxCNC internal) Off

axis

halui.axis.N.pos-commanded float out float out
Commanded axis position in machine coordinates

halui.axis.N.pos-feedback float out float out
Feedback axis position in machine coordinates

halui.axis.N.pos-relative float out float out
Commanded axis position in relative coordinates

home

halui.home-all bit in
pin for requesting home-all (only available when a valid homing sequence is specified)

SEE ALSO

HISTORY

BUGS

none known at this time.

AUTHOR

Written by Alex Joni, as part of the LinuxCNC project. Updated by John Thornton

REPORTING BUGS

Report bugs to alex_joni AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 Alex Joni.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

iocontrol − accepts NML I/O commands, interacts with HAL in userspace

SYNOPSIS

loadusr io [-ini *inifile***]**

DESCRIPTION

These pins are created by the userspace IO controller, usually found in $LINUXCNC_HOME/bin/io

The signals are turned on and off in userspace - if you have strict timing requirements or simply need more i/o, consider using the realtime synchronized i/o provided by **motion**(9) instead.

The inifile is searched for in the directory from which halcmd was run, unless an absolute path is specified.

PINS

iocontrol.0.coolant-flood
(Bit, Out) TRUE when flood coolant is requested

iocontrol.0.coolant-mist
(Bit, Out) TRUE when mist coolant is requested

iocontrol.0.emc-enable-in
(Bit, In) Should be driven FALSE when an external estop condition exists.

iocontrol.0.lube
(Bit, Out) TRUE when lube is requested

iocontrol.0.lube_level
(Bit, In) Should be driven FALSE when lubrication tank is empty.

iocontrol.0.tool-change
(Bit, Out) TRUE when a tool change is requested

iocontrol.0.tool-changed
(Bit, In) Should be driven TRUE when a tool change is completed.

iocontrol.0.tool-number
(s32, Out) Current tool number

iocontrol.0.tool-prep-number
(s32, Out) The number of the next tool, from the RS274NGC T-word

iocontrol.0.tool-prep-pocket
(s32, Out) The pocket number (location in tool storage mechanism) of the next tool, as described in the tool table

iocontrol.0.tool-prepare
(Bit, Out) TRUE when a T*n* tool prepare is requested

iocontrol.0.tool-prepared
(Bit, In) Should be driven TRUE when a tool prepare is completed.

iocontrol.0.user-enable-out
(Bit, Out) FALSE when an internal estop condition exists

iocontrol.0.user-request-enable
(Bit, Out) TRUE when the user has requested that estop be cleared

SEE ALSO

motion(9)

NAME

linuxcncrsh − text-mode interface for commanding LinuxCNC over the network

SYNOPSIS

linuxcncrsh [OPTIONS] [-- LINUXCNC_OPTIONS]

DESCRIPTION

linuxcncrsh is a user interface for LinuxCNC. Instead of popping up a GUI window like axis(1) and touchy(1) do, it processes text-mode commands that it receives via the network. A human (or a program) can interface with **linuxcncrsh** using telnet(1) or nc(1) or similar programs.

All features of LinuxCNC are available via the **linuxcncrsh** interface.

OPTIONS

-p,--port PORT_NUMBER

Specify the port for linuxcncrsh to listen on. Defaults to 5007 if omitted.

-n,--name SERVER_NAME

Sets the server name that linuxcncrsh will use to identify itself during handshaking with a new client. Defaults to EMCNETSVR if omitted.

-w,--connectpw PASSWORD

Specify the connection password to use during handshaking with a new client. Note that the password is sent in the clear, so it can be read by anyone who can read packets on the network between the server and the client. Defaults to EMC if omitted.

-e,--enablepw PASSWORD

Specify the password required to enable LinuxCNC via linuxcncrsh. Note that the password is sent in the clear, so it can be read by anyone who can read packets on the network between the server and the client. Defaults to EMCTOO if omitted.

-s,--sessions MAX_SESSIONS

Specify the maximum number of simultaneous connections. Defaults to -1 (no limit) if not specified.

In addition to the options listed above, linuxcncrsh accepts an optional special LINUXCNC_OPTION at the end:

-ini LINUXCNC_INI_FILE

LinuxCNC .ini file to use. The -ini option **must** be preceeded by two dashes: "--". Defaults to emc.ini if omitted.

Starting linuxcncrsh

To use linuxcncrsh instead of a normal LinuxCNC GUI like axis or touch, specify it in your .ini file like this:

[DISPLAY]

DISPLAY=linuxcncrsh

To use linuxcncrsh in addition to a normal GUI, you can either start it at the end of your .hal file, or run it by hand in a terminal window.

To start it from hal, add a line like this to the end of your .hal file:

loadusr linuxcncrsh [OPTIONS] [-- LINUXCNC_OPTIONS]

To start it from the terminal, run linuxcncrsh manually like this:

linuxcncrsh [OPTIONS] [-- LINUXCNC_OPTIONS]

Connecting

Once LinuxCNC is up and linuxcncrsh is running, you can connect to it using **telnet** or **nc** or similar:

telnet HOST PORT

HOST is the hostname or IP address of the computer running linuxcncrsh, and PORT is

the port it's listening on (5007 if you did not give linuxcncrsh the --port option).

Network protocol

linuxcncrsh accepts TCP connections on the port specified by the --port option, or 5007 if not specified.

The client sends requests, and the linuxcncrsh server returns replies. Requests consist of a command word followed by optional command-specific parameters. Requests and most request parameters are case insensitive. The exceptions are passwords, file paths and text strings.

Requests to linuxcncrsh are terminated with line endings, any combination of one or more '\r' and '\n' characters. Replies from linuxcncrsh are terminated with the sequence ´\r\n´.

The supported commands are as follows:

hello <password> <client> <version>

<password> must match linuxcncrsh's connect password, or "EMC" if no --connectpw was supplied. The three arguments may not contain whitespace. If a valid password was entered the server will respond with:

HELLO ACK <ServerName> <ServerVersion>

If an invalid password or any other syntax error occurs then the server responds with:

HELLO NAK

get <subcommand> [<parameters>]

The get command takes one of the LinuxCNC sub-commands (described in the section **LinuxCNC Subcommands**, below) and zero or more additional subcommand-specific parameters.

set <subcommand> <parameters>

The set command takes one of the LinuxCNC sub-commands (described in the section **LinuxCNC Subcommands**, below) and one or more additional parameters.

quit

The quit command disconnects the associated socket connection.

shutdown

The shutdown command tells LinuxCNC to shutdown and disconnect the session. This command may only be issued if the Hello has been successfully negotiated and the connection has control of the CNC (see **enable** subcommand in the **LinuxCNC Subcommands** section, below).

help

The help command will return help information in text format over the connection. If no parameters are specified, it will itemize the available commands. If a command is specified, it will provide usage information for the specified command. Help will respond regardless of whether a "Hello" has been successsfully negotiated.

LinuxCNC Subcommands

Subcommands for **get** and **set** are:

echo {on|off}

With get, any on/off parameter is ignored and the current echo state is returned. With set, sets the echo state as specified. Echo defaults to on when the connection is first established. When echo is on, all commands will be echoed upon receipt. This state is local to each connection.

verbose {on|off}

With get, any on/off parameter is ignored and the current verbose state is returned. With set, sets the verbose state as specified. When verbose mode is on, all set commands return positive acknowledgement in the form SET <COMMAND> ACK, and text error messages will be issued (FIXME: I don't know what this means). The verbose state is local to each connection, and starts out OFF on new connections.

enable {<passwd>|off}

The session's enable state indicates whether the current connection is enabled to perform control functions. With get, any parameter is ignored, and the current enable state is returned. With set

and a valid password matching linuxcncrsh's --enablepw (EMCTOO if not specified), the current connection is enabled for control functions. "OFF" may not be used as a password and disables control functions for this connection.

config [TBD]
Unused, ignore for now.

comm_mode {ascii|binary}
With get, any parameter is ignored and the current communications mode is returned. With set, will set the communications mode to the specified mode. The ascii mode is the text request/reply mode, the binary protocol is not currently designed or implemented.

comm_prot <version>
With get, any parameter is ignored and the current protocol version used by the server is returned. With set, sets the server to use the specified protocol version, provided it is lower than or equal to the highest version number supported by the server implementation.

inifile
Not currently implemented! With get, returns the string "emc.ini". Should return the full path and file name of the current configuration inifile. Setting this does nothing.

plat
With get, returns the string "Linux".

ini <var> <section>
Not currently implemented, do not use! Should return the string value of <var> in section <section> of the ini file.

debug <value>
With get, any parameter is ignored and the current integer value of EMC_DEBUG is returned. Note that the value of EMC_DEBUG returned is the from the UI's ini file, which may be different than emc's ini file. With set, sends a command to the EMC to set the new debug level, and sets the EMC_DEBUG global here to the same value. This will make the two values the same, since they really ought to be the same.

set_wait {none|received|done}
The set_wait setting controls the wait after receiving a command. It can be "none" (return right away), "received" (after the command was sent and received), or "done" (after the command was done). With get, any parameter is ignored and the current set_wait setting is returned. With set, set the set_wait setting to the specified value.

wait {received|done}
With set, force a wait for the previous command to be received, or done. This lets you wait in the event that "set_wait none" is in effect.

set_timeout <timeout>
With set, set the timeout for commands to return to <timeout> seconds. Timeout is a real number. If it's <= 0.0, it means wait forever. Default is 0.0, wait forever.

update {none|auto}
The update mode controls whether to return fresh or stale values for "get" requests. When the update mode is "none" it returns stale values, when it's "auto" it returns fresh values. Defaults to "auto" for new connections. Set this to "none" if you like to be confused.

error
With get, returns the current error string, or "ok" if no error.

operator_display
With get, returns the current operator display string, or "ok" if none.

operator_text
With get, returns the current operator text string, or "ok" if none.

time
With get, returns the time, in seconds, from the start of the epoch. This starting time depends on the platform.

estop {on|off}
With get, ignores any parameters and returns the current estop setting as "on" or "off". With set, sets the estop as specified. Estop "on" means the machine is in the estop state and won't run.

machine {on|off}
With get, ignores any parameters and returns the current machine power setting as "on" or "off". With set, sets the machine on or off as specified.

mode {manual|auto|mdi}
With get, ignores any parameters and returns the current machine mode. With set, sets the machine mode as specified.

mist {on|off}
With get, ignores any parameters and returns the current mist coolant setting. With set, sets the mist setting as specified.

flood {on|off}
With get, ignores any parameters and returns the current flood coolant setting. With set, sets the flood setting as specified.

lube {on|off}
With get, ignores any parameters and returns the current lube pump setting. With set, sets the lube pump setting as specified.

lube_level
With get, returns the lubricant level sensor reading as "ok" or "low". With set, mocks you for wishful thinking.

spindle {forward|reverse|increase|decrease|constant|off}
With get, any parameter is ignored and the current spindle state is returned as "forward", "reverse", "increase", "decrease", or "off". With set, sets the spindle as specified. Note that "increase" and "decrease" will cause a speed change in the corresponding direction until a "constant" command is sent.

brake {on|off}
With get, any parameter is ignored and the current brake setting is returned. With set, the brake is set as specified.

tool
With get, returns the id of the currently loaded tool.

tool_offset
With get, returns the currently applied tool length offset.

load_tool_table <file>
With set, loads the tool table specified by <file>.

home {0|1|2|...}
With set, homes the indicated axis.

jog_stop {0|1|2|...}
With set, stop any in-progress jog on the specified axis.

jog {0|1|2|...} <speed>
With set, jog the specified axis at <speed>; sign of speed is direction.

jog_incr {0|1|2|...} <speed> <incr>
With set, jog the indicated axis by increment <incr> at the <speed>; sign of speed is direction.

feed_override <percent>
With get, any parameter is ignored and the current feed override is returns (as a percentage of

commanded feed). With set, sets the feed override as specified.

spindle_override <percent>
With get, any parameter is ignored and the current spindle override is returnd (as a percentage of commanded speed). With set, sets the spindle override as specified.

abs_cmd_pos [{0|1|...}]
With get, returns the specified axis' commanded position in absolute coordinates. If no axis is specified, returns all axes' commanded absolute position.

abs_act_pos [{0|1|...}]
With get, returns the specified axis' actual position in absolute coordinates. If no axis is specified, returns all axes' actual absolute position.

rel_cmd_pos [{0|1|...}]
With get, returns the specified axis' commanded position in relative coordinates, including tool length offset. If no axis is specified, returns all axes' commanded relative position.

rel_act_pos [{0|1|...}]
With get, returns the specified axis' actual position in relative coordinates, including tool length offset. If no axis is specified, returns all axes' actual relative position.

joint_pos [{0|1|...}]
With get, returns the specified joint's actual position in absolute coordinates, excluding tool length offset. If no joint is specified, returns all joints' actual absolute position.

pos_offset [{X|Y|Z|R|P|W}]
With get, returns the position offset associated with the world coordinate provided.

joint_limit [{0|1|...}]
With get, returns limit status of the specified joint as "ok", "minsoft", "minhard", "maxsoft", or "maxhard". If no joint number is specified, returns the limit status of all joints.

joint_fault [{0|1|...}]
With get, returns the fault status of the specified joint as "ok" or "fault". If no joint number is specified, returns the fault status of all joints.

joint_homed [{0|1|...}]
With get, returns the homed status of the specified joint as "homed" or "not". If no joint number is specified, returns the homed status of all joints.

mdi <string>
With set, sends <string> as an MDI command.

task_plan_init
With set, initializes the program interpreter.

open <filename>
With set, opens the named file.

run [<StartLine>]
With set, runs the opened program. If no StartLine is specified, runs from the beginning. If a StartLine is specified, start line, runs from that line. A start line of -1 runs in verify mode.

pause
With set, pause program execution.

resume
With set, resume program execution.

abort
With set, abort program or MDI execution.

step
With set, step the program one line.

program
With get, returns the name of the currently opened program, or "none".

program_line
With get, returns the currently executing line of the program.

program_status
With get, returns "idle", "running", or "paused".

program_codes
With get, returns the string for the currently active program codes.

joint_type [<joint>]
With get, returns "linear", "angular", or "custom" for the type of the specified joint (or for all joints if none is specified).

joint_units [<joint>]
With get, returns "inch", "mm", "cm", or "deg", "rad", "grad", or "custom", for the corresponding native units of the specified joint (or for all joints if none is specified). The type of the axis (linear or angular) is used to resolve which type of units are returned. The units are obtained heuristically, based on the EMC_AXIS_STAT::units numerical value of user units per mm or deg. For linear joints, something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom". For angular joints, something close to 1.000 is deemed "deg", PI/180 is "rad", 100/90 is "grad", otherwise it's "custom".

program_units
Synonym for program_linear_units.

program_linear_units
With get, returns "inch", "mm", "cm", or "none", for the corresponding linear units that are active in the program interpreter.

program_angular_units
With get, returns "deg", "rad", "grad", or "none" for the corresponding angular units that are active in the program interpreter.

user_linear_units
With get, returns "inch", "mm", "cm", or "custom", for the corresponding native user linear units of the LinuxCNC trajectory level. This is obtained heuristically, based on the EMC_TRAJ_STAT::linearUnits numerical value of user units per mm. Something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom".

user_angular_units
Returns "deg", "rad", "grad", or "custom" for the corresponding native user angular units of the LinuxCNC trajectory level. Like with linear units, this is obtained heuristically.

display_linear_units
With get, returns "inch", "mm", "cm", or "custom", for the linear units that are active in the display. This is effectively the value of linearUnitConversion.

display_angular_units
With get, returns "deg", "rad", "grad", or "custom", for the angular units that are active in the display. This is effectively the value of angularUnitConversion.

linear_unit_conversion {inch|mm|cm|auto}
With get, any parameter is ignored and the active unit conversion is returned. With set, sets the unit to be displayed. If it's "auto", the units to be displayed match the program units.

angular_unit_conversion {deg|rad|grad|auto}
With get, any parameter is ignored and the active unit conversion is returned. With set, sets the units to be displayed. If it's "auto", the units to be displayed match the program units.

probe_clear
With set, clear the probe tripped flag.

probe_tripped
With get, return the probe state - has the probe tripped since the last clear?

probe_value
With get, return the current value of the probe signal.

probe
With set, move toward a certain location. If the probe is tripped on the way stop motion, record the position and raise the probe tripped flag.

teleop_enable [on|off]
With get, any parameter is ignored and the current teleop mode is returned. With set, sets the teleop mode as specified.

kinematics_type
With get, returns the type of kinematics functions used (identity=1, serial=2, parallel=3, custom=4).

override_limits {on|off}
With get, any parameter is ignored and the override_limits setting is returned. With set, the override_limits parameter is set as specified. If override_limits is on, disables end of travel hardware limits to allow jogging off of a limit. If parameters is off, then hardware limits are enabled.

optional_stop {0|1}
With get, any parameter is ignored and the current "optional stop on M1" setting is returned. With set, the setting is set as specified.

Example Session

This section shows an example session. Bold items are typed by you, non-bold is machine output.

The user connects to linuxcncrsh, handshakes with the server (hello), enables machine commanding from this session (set enable), brings the machine out of estop (set estop off) and turns it on (set machine on), homes all the axes, switches the machine to mdi mode, sends an MDI g-code command, then disconnects and shuts down LinuxCNC.

> **telnet localhost 5007**
Trying 127.0.0.1...
Connected to 127.0.0.1
Escape character is '^]'.
hello EMC user-typing-at-telnet 1.0
HELLO ACK EMCNETSVR 1.1
set enable EMCTOO
set enable EMCTOO
set mode manual
set mode manual
set estop off
set estop off
set machine on
set machine on
set home 0
set home 0
set home 1
set home 1
set home 2
set home 2
set mode mdi
set mode mdi
set mdi g0x1
set mdi g0x1
shutdown

shutdown
Connection closed by foreign host.

NAME

pyvcp − Virtual Control Panel for LinuxCNC

SYNOPSIS

pyvcp [-g *WxH+X+Y*] [-c *component-name*] *myfile.xml*

OPTIONS

-g *WxH+X+Y*

This sets the initial geometry of the root window. Use 'WxH' for just size, '+X+Y' for just position, or 'WxH+X+Y' for both. Size / position use pixel units. Position is referenced from top left.

-c *component-name*

Use *component-name* as the HAL component name. If the component name is not specified, the basename of the xml file is used.

SEE ALSO

Python Virtual Control Panel in the LinuxCNC documentation for a description of the xml syntax, along with examples

NAME

shuttlexpress − control HAL pins with the ShuttleXpress device made by Contour Design

SYNOPSIS

loadusr **shuttlexpress** *[DEVICE ...]*

DESCRIPTION

shuttlexpress is a userspace HAL component that interfaces Contour Design's ShuttleXpress device with LinuxCNC's HAL. The ShuttleXpress has five momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

If it is started without command-line arguments, it will probe all /dev/hidraw* device files for ShuttleXpress devices, and use all devices found. If it is started with command-line arguments, only will only probe the devices specified.

UDEV

The shuttlexpress module needs read permission on the /dev/hidraw* device files. This can be accomplished by adding a file **/etc/udev/rules.d/99-shuttlexpress.rules**, with the following contents:

SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"

A warning about the Jog Wheel

The ShuttleXpress device has an internal 8-bit counter for the current jog-wheel position. The shuttlexpress driver can not know this value until the ShuttleXpress device sends its first event. When the first event comes into the driver, the driver uses the device's reported jog-wheel position to initialize counts to 0.

This means that if the first event is generated by a jog-wheel move, that first move will be lost.

Any user interaction with the ShuttleXpress device will generate an event, informing the driver of the jog-wheel position. So if you (for example) push one of the buttons at startup, the jog-wheel will work fine and notice the first click.

Pins

(bit out) *shuttlexpress.0.button-0*

(bit out) *shuttlexpress.0.button-0-not*

(bit out) *shuttlexpress.0.button-1*

(bit out) *shuttlexpress.0.button-1-not*

(bit out) *shuttlexpress.0.button-2*

(bit out) *shuttlexpress.0.button-2-not*

(bit out) *shuttlexpress.0.button-3*

(bit out) *shuttlexpress.0.button-3-not*

(bit out) *shuttlexpress.0.button-4*

(bit out) *shuttlexpress.0.button-4-not*

The five buttons around the outside, starting with the counter-clockwise-most one.

(s32 out) *shuttlexpress.0.counts*

Accumulated counts from the jog wheel (the inner wheel).

(s32 out) *shuttlexpress.0.spring-wheel-s32*

The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, and ranges from -7 at the counter-clockwise extreme to +7 at the clockwise extreme.

(float out) *shuttlexpress.0.spring-wheel-f*

The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, -1 at the counter-clockwise extreme, and +1 at the clockwise extreme. (The ShuttleXpress device reports the spring-wheel position quantized from -7 to +7, so this pin reports only 15 discrete values in its range.)

NAME

hal − Introduction to the HAL API

DESCRIPTION

HAL stands for Hardware Abstraction Layer, and is used by LinuxCNC to transfer realtime data to and from I/O devices and other low-level modules.

hal.h defines the API and data structures used by the HAL. This file is included in both realtime and non-realtime HAL components. HAL uses the RTPAI real time interface, and the #define symbols RTAPI and ULAPI are used to distinguish between realtime and non-realtime code. The API defined in this file is implemented in hal_lib.c and can be compiled for linking to either realtime or user space HAL components.

The HAL is a very modular approach to the low level parts of a motion control system. The goal of the HAL is to allow a systems integrator to connect a group of software components together to meet whatever I/O requirements he (or she) needs. This includes realtime and non-realtime I/O, as well as basic motor control up to and including a PID position loop. What these functions have in common is that they all process signals. In general, a signal is a data item that is updated at regular intervals. For example, a PID loop gets position command and feedback signals, and produces a velocity command signal.

HAL is based on the approach used to design electronic circuits. In electronics, off-the-shelf components like integrated circuits are placed on a circuit board and their pins are interconnected to build whatever overall function is needed. The individual components may be as simple as an op-amp, or as complex as a digital signal processor. Each component can be individually tested, to make sure it works as designed. After the components are placed in a larger circuit, the signals connecting them can still be monitored for testing and troubleshooting.

Like electronic components, HAL components have pins, and the pins can be interconnected by signals.

In the HAL, a *signal* contains the actual data value that passes from one pin to another. When a signal is created, space is allocated for the data value. A *pin* on the other hand, is a pointer, not a data value. When a pin is connected to a signal, the pin's pointer is set to point at the signal's data value. This allows the component to access the signal with very little run-time overhead. (If a pin is not linked to any signal, the pointer points to a dummy location, so the realtime code doesn't have to deal with null pointers or treat unlinked variables as a special case in any way.)

There are three approaches to writing a HAL component. Those that do not require hard realtime performance can be written as a single user mode process. Components that need hard realtime performance but have simple configuration and init requirements can be done as a single kernel module, using either predefined init info, or insmod-time parameters. Finally, complex components may use both a kernel module for the realtime part, and a user space process to handle ini file access, user interface (possibly including GUI features), and other details.

HAL uses the RTAPI/ULAPI interface. If RTAPI is #defined hal_lib.c would generate a kernel module hal_lib.o that is insmoded and provides the functions for all kernel module based components. The same source file compiled with the ULAPI #define would make a user space hal_lib.o that is staticlly linked to user space code to make user space executables. The variable lists and link information are stored in a block of shared memory and protected with mutexes, so that kernel modules and any of several user mode programs can access the data.

REALTIME CONSIDERATIONS

For an explanation of realtime considerations, see **intro(3rtapi)**.

HAL STATUS CODES

Except as noted in specific manual pages, HAL returns negative errno values for errors, and nonnegative values for success.

SEE ALSO

intro(3rtapi)

NAME

hal_add_funct_to_thread − cause a function to be executed at regular intervals

SYNTAX

int hal_add_funct_to_thread(const char *_funct_name_, const char *_thread_name_,
int position)

int hal_del_funct_from_thread(const char *_funct_name_, const char *_thread_name_)

ARGUMENTS

funct_name

The name of the function

thread_name

The name of the thread

position

The desired location within the thread. This determines when the function will run, in relation to other functions in the thread. A positive number indicates the desired location as measured from the beginning of the thread, and a negative is measured from the end. So +1 means this function will become the first one to run, +5 means it will be the fifth one to run, -2 means it will be next to last, and -1 means it will be last. Zero is illegal.

DESCRIPTION

hal_add_funct_to_thread adds a function exported by a realtime HAL component to a realtime thread. This determines how often and in what order functions are executed.

hal_del_funct_from_thread removes a function from a thread.

RETURN VALUE

Returns a HAL status code.

REALTIME CONSIDERATIONS

Call only from realtime init code, not from user space or realtime code.

SEE ALSO

hal_thread_new(3hal), **hal_export_funct(3hal)**

NAME

hal_create_thread − Create a HAL thread

SYNTAX

int hal_create_thread(const char *_name_, unsigned long _period_, int _uses_fp_)

int hal_thread_delete(const char *_name_)

ARGUMENTS

name The name of the thread

period The interval, in nanoseconds, between iterations of the thread

uses_fp Must be nonzero if a function which uses floating-point will be attached to this thread.

DESCRIPTION

hal_create_thread establishes a realtime thread that will execute one or more HAL functions periodically.

All thread periods are rounded to integer multiples of the hardware timer period, and the timer period is based on the first thread created. Threads must be created in order, from the fastest to the slowest. HAL assigns decreasing priorities to threads that are created later, so creating them from fastest to slowest results in rate monotonic priority scheduling.

hal_delete_thread deletes a previously created thread.

REALTIME CONSIDERATIONS

Call only from realtime init code, not from user space or realtime code.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_export_funct(3hal)

NAME

hal_exit − Shut down HAL

SYNTAX

int hal_exit(int *comp_id*)

ARGUMENTS

comp_id

A HAL component identifier returned by an earlier call to **hal_init**.

DESCRIPTION

hal_exit shuts down and cleans up HAL and RTAPI. It must be called prior to exit by any module that called **hal_init**.

REALTIME CONSIDERATIONS

Call only from within user or init/cleanup code, not from realtime tasks.

RETURN VALUE

Returns a HAL status code.

NAME

hal_export_funct − create a realtime function callable from a thread

SYNTAX

typedef void(*hal_funct_t)(void * *arg*, long *period*)

int hal_export_funct(const char **name*, hal_funct_t *funct*, void **arg*, int *uses_fp*, int *reentrant*, int *comp_id*)

ARGUMENTS

name The name of the function.

funct The pointer to the function

arg The argument to be passed as the first parameter of *funct*

uses_fp Nonzero if the function uses floating-point operations, including assignment of floating point values with "=".

reentrant
If reentrant is non-zero, the function may be preempted and called again before the first call completes. Otherwise, it may only be added to one thread.

comp_id
A HAL component identifier returned by an earlier call to **hal_init**.

DESCRIPTION

hal_export_funct makes a realtime function provided by a component available to the system. A subsequent call to **hal_add_funct_to_thread** can be used to schedule the execution of the function as needed by the system.

When this function is placed on a HAL thread, and HAL threads are started, *funct* is called repeatedly with two arguments: *void *arg* is the same value that was given to **hal_export_funct**, and *long period* is the interval between calls in nanoseconds.

Each call to the function should do a small amount of work and return.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_create_thread(3hal), **hal_add_funct_to_thread(3hal)**

NAME

hal_init − Sets up HAL and RTAPI

SYNTAX

int hal_init(const char **modname*)

ARGUMENTS

modname

The name of this hal module

DESCRIPTION

hal_init sets up HAL and RTAPI. It must be called by any module that intends to use the API, before any other RTAPI calls.

modname can optionally point to a string that identifies the module. The string may be no longer than **RTAPI_NAME_LEN** characters. If *modname* is **NULL**, the system will assign a name.

REALTIME CONSIDERATIONS

Call only from within user or init/cleanup code, not from relatime tasks.

RETURN VALUE

On success, returns a positive integer module ID, which is used for subsequent calls to hal and rtapi APIs. On failure, returns a HAL error code.

NAME

hal_malloc − Allocate space in the HAL shared memory area

SYNTAX

void *hal_malloc(long int *size*)

ARGUMENTS

size Gives the size, in bytes, of the block

DESCRIPTION

hal_malloc allocates a block of memory from the main HAL shared memory area. It should be used by all components to allocate memory for HAL pins and parameters. It allocates ‘size’ bytes, and returns a pointer to the allocated space, or NULL (0) on error. The returned pointer will be properly aligned for any type HAL supports. A component should allocate during initialization all the memory it needs.

The allocator is very simple, and there is no ‘free’. The entire HAL shared memory area is freed when the last component calls **hal_exit**. This means that if you continuously install and remove one component while other components are present, you eventually will fill up the shared memory and an install will fail. Removing all components completely clears memory and you start fresh.

RETURN VALUE

A pointer to the allocated space, which is properly aligned for any variable HAL supports. Returns NULL on error.

NAME

hal_param_new − Create a HAL parameter

SYNTAX

int hal_param_bit_new(const char *_name_, hal_param_dir_t _dir_, hal_bit_t * _data_addr_, int _comp_id_)

int hal_param_float_new(const char *_name_, hal_param_dir_t _dir_, hal_float_t * _data_addr_, int _comp_id_)

int hal_param_u32_new(const char *_name_, hal_param_dir_t _dir_, hal_u32_t * _data_addr_, int _comp_id_)

int hal_param_s32_new(const char *_name_, hal_param_dir_t _dir_, hal_s32_t * _data_addr_, int _comp_id_)

int hal_param_bit_newf(hal_param_dir_t _dir_, hal_bit_t * _data_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_param_float_newf(hal_param_dir_t _dir_, hal_float_t * _data_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_param_u32_newf(hal_param_dir_t _dir_, hal_u32_t * _data_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_param_s32_newf(hal_param_dir_t _dir_, hal_s32_t * _data_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_param_new(const char *_name_, hal_type_t _type_, hal_in_dir_t _dir_, void *_data_addr_, int _comp_id_)

ARGUMENTS

name The name to give to the created parameter

dir The direction of the parameter, from the viewpoint of the component. It may be one of **HAL_RO**, or **HAL_RW** A component may assign a value to any parameter, but other programs (such as halcmd) may only assign a value to a parameter that is **HAL_RW**.

data_addr
The address of the data, which must lie within memory allocated by **hal_malloc**.

comp_id
: A HAL component identifier returned by an earlier call to **hal_init**.

fmt, ... A printf-style format string and arguments

type The type of the parameter, as specified in **hal_type_t(3hal)**.

DESCRIPTION

The **hal_param_new** family of functions create a new *param* object.

There are functions for each of the data types that the HAL supports. Pins may only be linked to signals of the same type.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_type_t(3hal)

NAME

funct − one-line description of funct

SYNTAX

#include "hal_parport.h"

int **hal_parport_get**(int *comp_id*, hal_parport_t **port*, unsigned short *base*, unsigned short *base_hi*, unsigned int *modes*)

void **hal_parport_release**(hal_parport_t **port*)

ARGUMENTS

comp_id
A HAL component identifier returned by an earlier call to **hal_init**.

port A pointer to a hal_parport_t structure

base The base address of the port (if port >= 16) or the linux port number of the port (if port < 16)

base_hi
The "high" address of the port (location of the ECP registers), 0 to use a probed high address, or -1 to disable the high address

modes Advise the driver of the desired port modes, from <linux/parport.h>. If a linux-detected port does not provide the requested modes, a warning is printed with rtapi_print_msg. This does not make the port request fail, because unfortunately, many systems that have working EPP parports are not detected as such by Linux.

DESCRIPTION

hal_parport_get allocates a parallel port for exclusive use of the named hal component. The port must be released with **hal_parport_release** before the component exits with **hal_exit**.

HIGH ADDRESS PROBING

If the port is a parallel port known to Linux, and Linux detected a high I/O address, this value is used. Otherwise, if base+0x400 is not registered to any device, it is used. Otherwise, no address is used. If no high address is detected, port->base_hi is 0.

PARPORT STRUCTURE

```
typedef struct
{
   unsigned short base;
   unsigned short base_hi;
   .... // and further unspecified fields
} hal_parport_t;
```

RETURN VALUE

hal_parport_get returns a HAL status code. On success, *port* is filled out with information about the allocated port. On failure, the contents of *port* are undefined except that it is safe (but not required) to pass this port to **hal_parport_release**.

hal_parport_release does not return a value. It always succeeds.

NAME

hal_pin_new − Create a HAL pin

SYNTAX

int hal_pin_bit_new(const char *_name_, hal_pin_dir_t _dir_, hal_bit_t ** _data_ptr_addr_, int _comp_id_)

int hal_pin_float_new(const char *_name_, hal_pin_dir_t _dir_, hal_float_t ** _data_ptr_addr_, int _comp_id_)

int hal_pin_u32_new(const char *_name_, hal_pin_dir_t _dir_, hal_u32_t ** _data_ptr_addr_, int _comp_id_)

int hal_pin_s32_new(const char *_name_, hal_pin_dir_t _dir_, hal_s32_t ** _data_ptr_addr_, int _comp_id_)

int hal_pin_bit_newf(hal_pin_dir_t _dir_, hal_bit_t ** _data_ptr_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_pin_float_newf(hal_pin_dir_t _dir_, hal_float_t ** _data_ptr_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_pin_u32_newf(hal_pin_dir_t _dir_, hal_u32_t ** _data_ptr_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_pin_s32_newf(hal_pin_dir_t _dir_, hal_s32_t ** _data_ptr_addr_, int _comp_id_, const char *_fmt_, ...)

int hal_pin_new(const char *_name_, hal_type_t _type_, hal_in_dir_t _dir_, void **_data_ptr_addr_, int _comp_id_)

ARGUMENTS

name The name of the pin

dir

The direction of the pin, from the viewpoint of the component. It may be one of **HAL_IN**, **HAL_OUT**, or **HAL_IO**. Any number of **HAL_IN** or **HAL_IO** pins may be connected to the same signal, but at most one **HAL_OUT** pin is permitted. A component may assign a value to a pin that is **HAL_OUT** or **HAL_IO**, but may not assign a value to a pin that is **HAL_IN**.

data_ptr_addr
The address of the pointer-to-data, which must lie within memory allocated by **hal_malloc**.

comp_id
A HAL component identifier returned by an earlier call to **hal_init**.

fmt,
A printf-style format string and arguments

type
The type of the param, as specified in **hal_type_t(3hal)**.

DESCRIPTION

The **hal_pin_new** family of functions create a new *pin* object. Once a pin has been created, it can be linked to a signal object using **hal_link**. A pin contains a pointer, and the component that owns the pin can dereference the pointer to access whatever signal is linked to the pin. (If no signal is linked, it points to a dummy signal.)

There are functions for each of the data types that the HAL supports. Pins may only be linked to signals of the same type.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_type_t(3hal), **hal_link(3hal)**

NAME

hal_ready − indicates that this component is ready

SYNTAX

hal_ready(int *comp_id*)

ARGUMENTS

comp_id

A HAL component identifier returned by an earlier call to **hal_init**.

DESCRIPTION

hal_ready indicates that this component is ready (has created all its pins, parameters, and functions). This must be called in any realtime HAL component before its **rtapi_app_init** exits, and in any userspace component before it enters its main loop.

RETURN VALUE

Returns a HAL status code.

NAME

hal_set_constructor − Set the constructor function for this component

SYNTAX

typedef int (*hal_constructor_t)(const char *prefix, const char *arg); int hal_set_constructor(int *comp_id*, hal_constructor_t *constructor*)

ARGUMENTS

comp_id A HAL component identifier returned by an earlier call to **hal_init**.

prefix The prefix to be given to the pins, parameters, and functions in the new instance

arg An argument that may be used by the component to customize this istance.

DESCRIPTION

As an experimental feature in HAL 2.1, components may be *constructable*. Such a component may create pins and parameters not only at the time the module is loaded, but it may create additional pins and parameters, and functions on demand.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

halcmd(1)

NAME

hal_set_lock, hal_get_lock − Set or get the HAL lock level

SYNTAX

int hal_set_lock(unsigned char *lock_type*)

int hal_get_lock()

ARGUMENTS

lock_type

The desired lock type, which may be a bitwise combination of: **HAL_LOCK_LOAD**, **HAL_LOCK_CONFIG**, **HAL_LOCK_PARAMS**, or **HAL_LOCK_PARAMS**. **HAL_LOCK_NONE** or 0 locks nothing, and **HAL_LOCK_ALL locks everything.**

DESCRIPTION

RETURN VALUE

hal_set_lock Returns a HAL status code. **hal_get_lock** returns the current HAL lock level or a HAL status code.

NAME

hal_signal_new, hal_signal_delete, hal_link, hal_unlink − Manipulate HAL signals

SYNTAX

int hal_signal_new(const char *_signal_name_, hal_type_t _type_)

int hal_signal_delete(const char *_signal_name_)

int hal_link(const char *_pin_name_, const char *_signal_name_)

int hal_unlink(const char *_pin_name_)

ARGUMENTS

signal_name
: The name of the signal

pin_name
: The name of the pin

type The type of the signal, as specified in **hal_type_t(3hal)**.

DESCRIPTION

hal_signal_new creates a new signal object. Once a signal has been created, pins can be linked to it with **hal_link**. The signal object contains the actual storage for the signal data. Pin objects linked to the signal have pointers that point to the data. 'name' is the name of the new signal. It may be no longer than HAL_NAME_LEN characters. If there is already a signal with the same name the call will fail.

hal_link links a pin to a signal. If the pin is already linked to the desired signal, the command succeeds. If the pin is already linked to some other signal, it is an error. In either case, the existing connection is not modified. (Use 'hal_unlink' to break an existing connection.) If the signal already has other pins linked to it, they are unaffected - one signal can be linked to many pins, but a pin can be linked to only one signal.

hal_unlink unlinks any signal from the specified pin.

hal_signal_delete deletes a signal object. Any pins linked to the object are unlinked.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_type_t(3hal)

NAME

hal_start_threads − Allow HAL threads to begin executing

SYNTAX

int hal_start_threads()

int hal_stop_threads()

ARGUMENTS

DESCRIPTION

hal_start_threads starts all threads that have been created. This is the point at which realtime functions start being called.

hal_stop_threads stops all threads that were previously started by **hal_start_threads**. It should be called before any component that is part of a system exits.

RETURN VALUE

Returns a HAL status code.

SEE ALSO

hal_export_funct(3hal), **hal_create_thread(3hal)**, **hal_add_funct_to_thread(3hal)**

NAME

hal_type_t − typedefs for HAL datatypes

DESRCIPTION

typedef ... **hal_bit_t;**

A volatile type which may have a value of 0 or nonzero.

typedef ... **hal_s32_t**;

A volatile type which may have a value from -2147483648 to 2147483647.

typedef ... **hal_u32_t**;

A volatile type which may have a value from 0 to 4294967295.

typedef ... **hal_float_t**;

A volatile floating-point type, which typically has the same precision and range as the C type **double**.

typedef ... **real_t**;

A nonvolatile floating-point type with at least as much precision as **hal_float_t**.

typedef ... **ireal_t**;

A nonvolatile unsigned integral type the same size as **hal_float_t**.

typedef enum **hal_type_t**;

HAL_BIT

Corresponds to the type **hal_bit_t**.

HAL_FLOAT

Corresponds to the type **hal_float_t**.

HAL_S32

Corresponds to the type **hal_s32_t**.

HAL_U32

Corresponds to the type **hal_u32_t**.

NOTES

hal_bit_t is typically a typedef to an integer type whose range is larger than just 0 and 1. When testing the value of a **hal_bit_t**, never compare it to 1. Prefer one of the following:

- if(b)
- if(b != 0)

It is often useful to refer to a type that can represent all the values as a hal type, but without the volatile qualifier. The following types correspond with the hal types:

hal_bit_t	int
hal_s32_t	__s32
hal_u32_t	__u32
hal_float_t	hal_real_t

Take care not to use the types **s32** and **u32**. These will compile in kernel modules but not in userspace, and not for "realtime components" when using simulated (userspace) realtime.

SEE ALSO

hal_pin_new(3hal), **hal_param_new(3hal)**

NAME

undocumented − undocumented functions in HAL

SEE ALSO

The header file *hal.h*. Most hal functions have documentation in that file.

NAME

rtapi − Introduction to the RTAPI API

DESCRIPTION

RTAPI is a library providing a uniform API for several real time operating systems. As of ver 2.1, RTLinux, RTAI, and a pure userspace simulator are supported.

HEADER FILES

rtapi.h

The file **rtapi.h** defines the RTAPI for both realtime and non-realtime code. This is a change from Rev 2, where the non-realtime (user space) API was defined in ulapi.h and used different function names. The symbols RTAPI and ULAPI are used to determine which mode is being compiled, RTAPI for realtime and ULAPI for non-realtime.

rtapi_math.h

The file rtapi_math.h defines floating-point functions and constants. It should be used instead of <math.h> in rtapi real-time components.

rtapi_string.h

The file rtapi_string.h defines string-related functions. It should be used instead of <string.h> in rtapi real-time components.

REALTIME CONSIDERATIONS

Userspace code

Certain functions are not available in userspace code. This includes functions that perform direct device access such as **rtapi_inb(3)**.

Init/cleanup code

Certain functions may only be called from realtime init/cleanup code. This includes functions that perform memory allocation, such as **rtapi_shmem_new(3)**.

Realtime code

Only a few functions may be called from realtime code. This includes functions that perform direct device access such as **rtapi_inb(3)**. It excludes most Linux kernel APIs such as do_gettimeofday(3) and many rtapi APIs such as rtapi_shmem_new(3).

Simulator

For an RTAPI module to be buildable in the "sim" environment (fake realtime system without special privileges), it must not use **any** linux kernel APIs, and must not use the RTAPI APIs for direct device access such as **rtapi_inb(3)**. This automatically includes any hardware device drivers, and also devices which use Linux kernel APIs to do things like create special devices or entries in the **/proc** filesystem.

RTAPI STATUS CODES

Except as noted in specific manual pages, RTAPI returns negative errno values for errors, and nonnegative values for success.

NAME

rtapi_app_exit − User-provided function to shut down a component

SYNTAX

void **rtapi_app_exit**(void) { ... }

ARGUMENTS

None

DESCRIPTION

The body of **rtapi_app_exit**, which is provided by the component author, generally consists of a call to rtapi_exit or hal_exit, preceded by other component-specific shutdown code.

This code is called when unloading a component which successfully initialized (i.e., returned zero from its **rtapi_app_main**). It is not called when the component did not successfully initialize.

RETURN CODE

None.

REALTIME CONSIDERATIONS

Called automatically by the rtapi infrastructure in an initialization (not realtime) context.

SEE ALSO

rtapi_app_main(3rtapi), **rtapi_exit(3rtapi)**, **hal_exit(3hal)**

NAME

rtapi_app_main − User-provided function to initialize a component

SYNTAX

#include "rtapi_app.h" int **rtapi_app_main**(void) { ... }

ARGUMENTS

None

DESCRIPTION

The body of **rtapi_app_main**, which is provided by the component author, generally consists of a call to rtapi_init or hal_init, followed by other component-specific initialization code.

RETURN VALUE

Return 0 for success. Return a negative errno value (e.g., -EINVAL) on error. Existing code also returns RTAPI or HAL error values, but using negative errno values gives better diagnostics from insmod.

REALTIME CONSIDERATIONS

Called automatically by the rtapi infrastructure in an initialization (not realtime) context.

SEE ALSO

rtapi_app_exit(3rtapi), **rtapi_init(3rtapi)**, **hal_init(3hal)**

NAME

rtapi_clock_set_period − set the basic time interval for realtime tasks

SYNTAX

rtapi_clock_set_period(long int *nsec*)

ARGUMENTS

nsec The desired basic time interval for realtime tasks.

DESCRIPTION

rtapi_clock_set_period sets the basic time interval for realtime tasks. All periodic tasks will run at an integer multiple of this period. The first call to **rtapi_clock_set_period** with *nsec* greater than zero will start the clock, using *nsec* as the clock period in nano-seconds. Due to hardware and RTOS limitations, the actual period may not be exactly what was requested. On success, the function will return the actual clock period if it is available, otherwise it returns the requested period. If the requested period is outside the limits imposed by the hardware or RTOS, it returns **-EINVAL** and does not start the clock. Once the clock is started, subsequent calls with non-zero *nsec* return **-EINVAL** and have no effect. Calling **rtapi_clock_set_period** with *nsec* set to zero queries the clock, returning the current clock period, or zero if the clock has not yet been started.

REALTIME CONSIDERATIONS

Call only from within init/cleanup code, not from realtime tasks. This function is not available from user (non-realtime) code.

RETURN VALUE

The actual period provided by the RTOS, which may be different than the requested period, or a RTAPI status code.

NAME

rtapi_delay − Busy-loop for short delays

SYNTAX

void rtapi_delay(long int *nsec*)

void rtapi_delay_max()

ARGUMENTS

nsec The desired delay length in nanoseconds

DESCRIPTION

rtapi_delay is a simple delay. It is intended only for short delays, since it simply loops, wasting CPU cycles.

rtapi_delay_max returns the max delay permitted (usually approximately 1/4 of the clock period). Any call to **rtapi_delay** requesting a delay longer than the max will delay for the max time only.

rtapi_delay_max should be called before using **rtapi_delay** to make sure the required delays can be achieved. The actual resolution of the delay may be as good as one nano-second, or as bad as a several microseconds.

REALTIME CONSIDERATIONS

May be called from init/cleanup code, and from within realtime tasks.

RETURN VALUE

rtapi_delay_max returns the maximum delay permitted.

SEE ALSO

rtapi_clock_set_period(3rtapi)

NAME

rtapi_exit − Shut down RTAPI

SYNTAX

int rtapi_exit(int *module_id*)

ARGUMENTS

module_id

An rtapi module identifier returned by an earlier call to **rtapi_init**.

DESCRIPTION

rtapi_exit shuts down and cleans up the RTAPI. It must be called prior to exit by any module that called **rtapi_init**.

REALTIME CONSIDERATIONS

Call only from within user or init/cleanup code, not from relatime tasks.

RETURN VALUE

Returns a RTAPI status code.

NAME

rtapi_get_time − get the current time

SYNTAX

long long rtapi_get_time()

long long rtapi_get_clocks()

DESCRIPTION

rtapi_get_time returns the current time in nanoseconds. Depending on the RTOS, this may be time since boot, or time since the clock period was set, or some other time. Its absolute value means nothing, but it is monotonically increasing and can be used to schedule future events, or to time the duration of some activity. Returns a 64 bit value. The resolution of the returned value may be as good as one nano-second, or as poor as several microseconds. May be called from init/cleanup code, and from within realtime tasks.

rtapi_get_clocks returns the current time in CPU clocks. It is fast, since it just reads the TSC in the CPU instead of calling a kernel or RTOS function. Of course, times measured in CPU clocks are not as convenient, but for relative measurements this works fine. Its absolute value means nothing, but it is monotonically increasing and can be used to schedule future events, or to time the duration of some activity. (on SMP machines, the two TSC's may get out of sync, so if a task reads the TSC, gets swapped to the other CPU, and reads again, the value may decrease. RTAPI tries to force all RT tasks to run on one CPU.) Returns a 64 bit value. The resolution of the returned value is one CPU clock, which is usually a few nanoseconds to a fraction of a nanosecond.

Note that *long long* math may be poorly supported on some platforms, especially in kernel space. Also note that rtapi_print() will NOT print *long long*s. Most time measurements are relative, and should be done like this:

```
deltat = (long int)(end_time - start_time);
```

where end_time and start_time are longlong values returned from rtapi_get_time, and deltat is an ordinary long int (32 bits). This will work for times up to a second or so, depending on the CPU clock frequency. It is best used for millisecond and microsecond scale measurements though.

RETURN VALUE

Returns the current time in nanoseconds or CPU clocks.

NOTES

Certain versions of the Linux kernel provide a global variable **cpu_khz**. Computing

```
deltat = (end_clocks - start_clocks) / cpu_khz:
```

gives the duration measured in milliseconds. Computing

```
deltat = (end_clocks - start_clocks) * 1000000 / cpu_khz:
```

gives the duration measured in nanoseconds for deltas less than about 9 trillion clocks (e.g., 3000 seconds at 3GHz).

REALTIME CONSIDERATIONS

May be called from init/cleanup code and from within realtime tasks. Not available in userspace components.

NAME

rtapi_init − Sets up RTAPI

SYNTAX

int rtapi_init(const char **modname*)

ARGUMENTS

modname

The name of this rtapi module

DESCRIPTION

rtapi_init sets up the RTAPI. It must be called by any module that intends to use the API, before any other RTAPI calls.

modname can optionally point to a string that identifies the module. The string will be truncated at **RTAPI_NAME_LEN** characters. If *modname* is **NULL**, the system will assign a name.

REALTIME CONSIDERATIONS

Call only from within user or init/cleanup code, not from relatime tasks.

RETURN VALUE

On success, returns a positive integer module ID, which is used for subsequent calls to rtapi_xxx_new, rtapi_xxx_delete, and rtapi_exit. On failure, returns an RTAPI error code.

NAME

rtapi_module_param − Specifying module parameters

SYNTAX

RTAPI_MP_INT(*var*, *description*)

RTAPI_MP_LONG(*var*, *description*)

RTAPI_MP_STRING(*var*, *description*)

RTAPI_MP_ARRAY_INT(*var*, *num*, *description*)

RTAPI_MP_ARRAY_LONG(*var*, *num*, *description*)

RTAPI_MP_ARRAY_STRING(*var*, *num*, *description*)

MODULE_LICENSE(*license*)

MODULE_AUTHOR(*author*)

MODULE_DESCRIPTION(*description*)

EXPORT_FUNCTION(*function*)

ARGUMENTS

var The variable where the parameter should be stored

description
A short description of the parameter or module

num The maximum number of values for an array parameter

license The license of the module, for instance "GPL"

author The author of the module

function
The pointer to the function to be exported

DESCRIPTION

These macros are portable ways to declare kernel module parameters. They must be used in the global scope, and are not followed by a terminating semicolon. They must be used after the associated variable or function has been defined.

NOTES

EXPORT_FUNCTION makes a symbol available for use by a subsequently loaded component. It is unrelated to hal functions, which are described in hal_export_funct(3hal)

Interpretation of license strings

MODULE_LICENSE follows the kernel's definition of license strings. Notably, "GPL" indicates "GNU Public License v2 *or later*". (emphasis ours).

"GPL"
GNU Public License v2 or later

"GPL v2"
GNU Public License v2

"GPL and additional rights"
GNU Public License v2 rights and more

"Dual BSD/GPL"
GNU Public License v2 or BSD license choice

"Dual MIT/GPL"
GNU Public License v2 or MIT license choice

"Dual MPL/GPL"
GNU Public License v2 or Mozilla license choice

"Proprietary"
Non-free products

It is still good practice to include a license block which indicates the author, copyright date, and disclaimer of warranty as recommended by the GNU GPL.

REALTIME CONSIDERATIONS

Not available in userspace code.

NAME

rtapi_mutex − Mutex-related functions

SYNTAX

int rtapi_mutex_try(unsigned long *_mutex_)

void rtapi_mutex_get(unsigned long *_mutex_)

void rtapi_mutex_give(unsigned long *_mutex_)

ARGUMENTS

mutex A pointer to the mutex.

DESCRIPTION

rtapi_mutex_try makes a non-blocking attempt to get the mutex. If the mutex is available, it returns 0, and the mutex is no longer available. Otherwise, it returns a nonzero value.

rtapi_mutex_get blocks until the mutex is available.

rtapi_mutex_give releases a mutex acquired by **rtapi_mutex_try** or **rtapi_mutex_get**.

REALTIME CONSIDERATIONS

rtapi_mutex_give and **rtapi_mutex_try** may be used from user, init/cleanup, and realtime code.

rtapi_mutex_get may not be used from realtime code.

RETURN VALUE

rtapi_mutex_try returns 0 for if the mutex was claimed, and nonzero otherwise.

rtapi_mutex_get and **rtapi_mutex_gif** have no return value.

NAME

rtapi_outb, rtapi_inb − Perform hardware I/O

SYNTAX

void rtapi_outb(unsigned char *byte*, unsigned int *port*)

unsigned char rtapi_inb(unsigned int *port*)

ARGUMENTS

port The address of the I/O port

byte The byte to be written to the port

DESCRIPTION

rtapi_outb writes a byte to a hardware I/O port. **rtapi_inb** reads a byte from a hardware I/O port.

REALTIME CONSIDERATIONS

May be called from init/cleanup code and from within realtime tasks. Not available in userspace components.

RETURN VALUE

rtapi_inb returns the byte read from the given I/O port

NOTES

The I/O address should be within a region previously allocated by **rtapi_request_region**. Otherwise, another real-time module or the Linux kernel might attempt to access the I/O region at the same time.

SEE ALSO

rtapi_region(3rtapi)

NAME

rtapi_print, rtapi_print_msg − print diagnostic messages

SYNTAX

void rtapi_print(const char *_fmt_, ...)

void rtapi_print_msg(int level, const char *_fmt_, ...)

typedef void(***rtapi_msg_handler_t**)(msg_level_t *level*, const char **msg*);

void **rtapi_set_msg_handler**(rtapi_msg_handler_t *handler*);

rtapi_msg_handler_t **rtapi_set_msg_handler**(void);

ARGUMENTS

level A message level: One of **RTAPI_MSG_ERR**, **RTAPI_MSG_WARN**, **RTAPI_MSG_INFO**, or **RTAPI_MSG_DBG**.

handler
A function to call from **rtapi_print** or **rtapi_print_msg** to actually output the message.

fmt

... Other arguments are as for *printf(3)*.

DESCRIPTION

rtapi_print and **rtapi_print_msg** work like the standard C printf functions, except that a reduced set of formatting operations are supported.

Depending on the RTOS, the default may be to print the message to stdout, stderr, a kernel log, etc. In RTAPI code, the action may be changed by a call to **rtapi_set_msg_handler**. A **NULL** argument to **rtapi_set_msg_handler** restores the default handler. **rtapi_msg_get_handler** returns the current handler. When the message came from **rtapi_print**, *level* is RTAPI_MSG_ALL.

rtapi_print_msg works like rtapi_print but only prints if *level* is less than or equal to the current message level.

REALTIME CONSIDERATIONS

rtapi_print and **rtapi_print_msg** May be called from user, init/cleanup, and realtime code. **rtapi_get_msg_handler** and **ftapi_set_msg_handler** may be called from realtime init/cleanup code. A message handler passed to **rtapi_set_msg_handler** may only call functions that can be called from real-time code.

RETURN VALUE

None.

SEE ALSO

rtapi_set_msg_level(3rtapi), **rtapi_get_msg_level(3rtapi)**, **printf(3)**

NAME

rtapi_prio − thread priority functions

SYNTAX

int rtapi_prio_highest()

int rtapi_prio_lowest()

int rtapi_prio_next_higher(int *prio*)

int rtapi_prio_next_lower(int *prio*)

ARGUMENTS

prio A value returned by a prior **rtapi_prio_xxx** call

DESCRIPTION

The **rtapi_prio_xxxx** functions provide a portable way to set task priority. The mapping of actual priority to priority number depends on the RTOS. Priorities range from **rtapi_prio_lowest** to **rtapi_prio_highest**, inclusive. To use this API, use one of two methods:

1) Set your lowest priority task to **rtapi_prio_lowest**, and for each task of the next lowest priority, set their priorities to **rtapi_prio_next_higher(previous)**.

2) Set your highest priority task to **rtapi_prio_highest**, and for each task of the next highest priority, set their priorities to **rtapi_prio_next_lower(previous)**.

N.B. A high priority task will pre-empt or interrupt a lower priority task. Linux is always the lowest priority!

REALTIME CONSIDERATIONS

Call these functions only from within init/cleanup code, not from realtime tasks.

RETURN VALUE

Returns an opaque real-time priority number.

SEE ALSO

rtapi_task_new(3rtapi)

NAME

rtapi_region − functions to manage I/O memory regions

SYNTAX

void *rtapi_request_region(unsigned long *base*, unsigned long int *size*, const char **name*)

void rtapi_release_region(unsigned long *base*, unsigned long int *size*)

ARGUMENTS

base The base address of the I/O region

size The size of the I/O region

name The name to be shown in /proc/ioports

DESCRIPTION

rtapi_request_region reserves I/O memory starting at *base* and going for *size* bytes.

REALTIME CONSIDERATIONS

May be called from realtime init/cleanup code only.

BUGS

On kernels before 2.4.0, **rtapi_request_region** always suceeds.

RETURN VALUE

rtapi_request_region returns NULL if the allocation fails, and a non-NULL value otherwise.

rtapi_release_region has no return value.

NAME

rtapi_get_msg_level, rtapi_set_msg_level − Get or set the logging level

SYNTAX

int rtapi_set_msg_level(int *level*)

int rtapi_get_msg_level()

ARGUMENTS

level The desired logging level

DESCRIPTION

Get or set the RTAPI message level used by **rtapi_print_msg**. Depending on the RTOS, this level may apply to a single RTAPI module, or it may apply to a group of modules.

REALTIME CONSIDERATIONS

May be called from user, init/cleanup, and realtime code.

RETURN VALUE

rtapi_set_msg_level returns a status code, and **rtapi_get_msg_level** returns the current level.

SEE ALSO

rtapi_print_msg(3rtapi)

NAME

rtapi_shmem − Functions for managing shared memory blocks

SYNTAX

int rtapi_shmem_new(int *key*, int *module_id*, unsigned long int *size*)

int rtapi_shmem_delete(int *shmem_id*, int *module_id*)

int rtapi_shmem_getptr(int *shmem_id*, void ** *ptr*)

ARGUMENTS

key Identifies the memory block. Key must be nonzero. All modules wishing to use the same memory must use the same key.

module_id
Module identifier returned by a prior call to **rtapi_init**.

size The desired size of the shared memory block, in bytes

ptr The pointer to the shared memory block. Note that the block may be mapped at a different address for different modules.

DESCRIPTION

rtapi_shmem_new allocates a block of shared memory. *key* identifies the memory block, and must be nonzero. All modules wishing to access the same memory must use the same key. *module_id* is the ID of the module that is making the call (see rtapi_init). The block will be at least *size* bytes, and may be rounded up. Allocating many small blocks may be very wasteful. When a particular block is allocated for the first time, the first 4 bytes are zeroed. Subsequent allocations of the same block by other modules or processes will not touch the contents of the block. Applications can use those bytes to see if they need to initialize the block, or if another module already did so. On success, it returns a positive integer ID, which is used for all subsequent calls dealing with the block. On failure it returns a negative error code.

rtapi_shmem_delete frees the shared memory block associated with *shmem_id*. *module_id* is the ID of the calling module. Returns a status code.

rtapi_shmem_getptr sets **ptr* to point to shared memory block associated with *shmem_id*.

REALTIME CONSIDERATIONS

rtapi_shmem_getptr may be called from user code, init/cleanup code, or realtime tasks.

rtapi_shmem_new and **rtapi_shmem_dete** may not be called from realtime tasks.

RETURN VALUE

NAME

rtapi_snprintf, rtapi_vsnprintf − Perform snprintf-like string formatting

SYNTAX

int rtapi_snprintf(char **buf*, unsigned long int *size*, const char **fmt*, ...)

int rtapi_vsnprintf(char **buf*, unsigned long int *size*, const char **fmt*, va_list *apfB)*

ARGUMENTS

As for *snprintf(3)* or *vsnprintf(3)*.

DESCRIPTION

These functions work like the standard C printf functions, except that a reduced set of formatting operations are supported.

REALTIME CONSIDERATIONS

May be called from user, init/cleanup, and realtime code.

RETURN VALUE

The number of characters written to *buf*.

SEE ALSO

printf(3)

NAME

rtapi_task_new − create a realtime task

SYNTAX

int rtapi_task_new(void (**taskcode*)(void*), void **arg*, int *prio*, unsigned long *stacksize*, int *uses_fp*)

int rtapi_task_delete(int *task_id*)

ARGUMENTS

taskcode
: A pointer to the function to be called when the task is started

arg
: An argument to be passed to the *taskcode* function when the task is started

prio
: A task priority value returned by **rtapi_prio_xxxx**

uses_fp
: A flag that tells the OS whether the task uses floating point or not.

task_id
: A task ID returned by a previous call to **rtapi_task_new**

DESCRIPTION

rtapi_task_new creates but does not start a realtime task. The task is created in the "paused" state. To start it, call either **rtapi_task_start** for periodic tasks, or **rtapi_task_resume** for free-running tasks.

REALTIME CONSIDERATIONS

Call only from within init/cleanup code, not from realtime tasks.

RETURN VALUE

On success, returns a positive integer task ID. This ID is used for all subsequent calls that need to act on the task. On failure, returns an RTAPI status code.

SEE ALSO

rtapi_prio(3rtapi), **rtapi_task_start(3rtapi)**, **rtapi_task_wait(3rtapi)**, **rtapi_task_resume(3rtapi)**

NAME

rtapi_task_pause, rtapi_task_resume − pause and resume real-time tasks

SYNTAX

void rtapi_task_pause(int *task_id*)

void rtapi_task_resume(int *task_id*)

ARGUMENTS

task_id An RTAPI task identifier returned by an earlier call to **rtapi_task_new**.

DESCRIPTION

rtapi_task_resume starts a task in free-running mode. The task must be in the "paused" state.

A free running task runs continuously until either:

1) It is prempted by a higher priority task. It will resume as soon as the higher priority task releases the CPU.

2) It calls a blocking function, like **rtapi_sem_take**. It will resume when the function unblocks.

3) It is returned to the "paused" state by **rtapi_task_pause**. May be called from init/cleanup code, and from within realtime tasks.

rtapi_task_pause causes a task to stop execution and change to the "paused" state. The task can be free-running or periodic. Note that **rtapi_task_pause** may called from any task, or from init or cleanup code, not just from the task that is to be paused. The task will resume execution when either **rtapi_task_resume** or **rtapi_task_start** (depending on whether this is a free-running or periodic task) is called.

REALTIME CONSIDERATIONS

May be called from init/cleanup code, and from within realtime tasks.

RETURN VALUE

An RTAPI status code.

SEE ALSO

rtapi_task_new(3rtapi), **rtapi_task_start(3rtapi)**

NAME

rtapi_task_start − start a realtime task in periodic mode

SYNTAX

int rtapi_task_start(int *task_id*, unsigned long *period_nsec*)

ARGUMENTS

task_id A task ID returned by a previous call to **rtapi_task_new**

period_nsec

The clock period in nanoseconds between iterations of a periodic task

DESCRIPTION

rtapi_task_start starts a task in periodic mode. The task must be in the *paused* state.

REALTIME CONSIDERATIONS

Call only from within init/cleanup code, not from realtime tasks.

RETURN VALUE

Returns an RTAPI status code.

SEE ALSO

rtapi_task_new(3rtapi), **rtapi_task_pause(3rtapi)**, **rtapi_task_resume(3rtapi)**

NAME

rtapi_task_wait − suspend execution of this periodic task

SYNTAX

void rtapi_task_wait()

DESCRIPTION

rtapi_task_wait suspends execution of the current task until the next period. The task must be periodic. If not, the result is undefined.

REALTIME CONSIDERATIONS

Call only from within a periodic realtime task

RETURN VALUE

None

SEE ALSO

rtapi_task_start(3rtapi), **rtapi_task_pause(3rtapi)**

NAME

undocumented − undocumented functions in RTAPI

SEE ALSO

The header file *rtapi.h*. Most rtapi functions have documentation in that file.

NAME

abs − Compute the absolute value and sign of the input signal

SYNOPSIS

loadrt abs [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

abs.*N* (requires a floating-point thread)

PINS

abs.*N***.in** float in
Analog input value

abs.*N***.out** float out
Analog output value, always positive

abs.*N***.sign** bit out
Sign of input, false for positive, true for negative

abs.*N***.is-positive** bit out
TRUE if input is positive, FALSE if input is 0 or negative

abs.*N***.is-negative** bit out
TRUE if input is negative, FALSE if input is 0 or positive

LICENSE

GPL

NAME

abs_s32 − Compute the absolute value and sign of the input signal

SYNOPSIS

loadrt abs_s32 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

abs-s32.*N*

PINS

abs-s32.*N***.in** s32 in
input value

abs-s32.*N***.out** s32 out
output value, always non-negative

abs-s32.*N***.sign** bit out
Sign of input, false for positive, true for negative

abs-s32.*N***.is-positive** bit out
TRUE if input is positive, FALSE if input is 0 or negative

abs-s32.*N***.is-negative** bit out
TRUE if input is negative, FALSE if input is 0 or positive

LICENSE

GPL

NAME

and2 − Two-input AND gate

SYNOPSIS

loadrt and2 [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

and2.*N*

PINS

and2.*N***.in0** bit in
and2.*N***.in1** bit in
and2.*N***.out** bit out

out is computed from the value of **in0** and **in1** according to the following rule:

in0=TRUE in1=TRUE
out=TRUE

Otherwise,
out=FALSE

LICENSE

GPL

NAME

at_pid − proportional/integral/derivative controller with auto tuning

SYNOPSIS

loadrt at_pid [num_chan=*num* **| names=***name1***[,***name2...***]]**

DESCRIPTION

at_pid is a classic Proportional/Integral/Derivative controller, used to control position or speed feedback loops for servo motors and other closed-loop applications.

at_pid supports a maximum of sixteen controllers. The number that are actually loaded is set by the **num_chan** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is three.

If **debug** is set to 1 (the default is 0), some additional HAL parameters will be exported, which might be useful for tuning, but are otherwise unnecessary.

at_pid has a built in auto tune mode. It works by setting up a limit cycle to characterize the process. From this, **Pgain/Igain/Dgain** or **Pgain/Igain/FF1** can be determined using Ziegler-Nichols. When using **FF1**, scaling must be set so that **output** is in user units per second.

During auto tuning, the **command** input should not change. The limit cycle is setup around the commanded position. No initial tuning values are required to start auto tuning. Only **tune-cycles**, **tune-effort** and **tune-mode** need be set before starting auto tuning. When auto tuning completes, the tuning parameters will be set. If running from LinuxCNC, the FERROR setting for the axis being tuned may need to be loosened up as it must be larger than the limit cycle amplitude in order to avoid a following error.

To perform auto tuning, take the following steps. Move the axis to be tuned, to somewhere near the center of it's travel. Set **tune-cycles** (the default value should be fine in most cases) and **tune-mode**. Set **tune-effort** to a small value. Set **enable** to true. Set **tune-mode** to true. Set **tune-start** to true. If no oscillation occurs, or the oscillation is too small, slowly increase **tune-effort**. Auto tuning can be aborted at any time by setting **enable** or **tune-mode** to false.

NAMING

The names for pins, parameters, and functions are prefixed as:
pid.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **pid.N.** format is shown in the following descriptions.

FUNCTIONS

pid.*N*.do-pid-calcs (uses floating-point)
Does the PID calculations for control loop *N*.

PINS

pid.*N*.command float in
The desired (commanded) value for the control loop.

pid.*N*.feedback float in
The actual (feedback) value, from some sensor such as an encoder.

pid.*N*.error float out
The difference between command and feedback.

pid.*N*.output float out
The output of the PID loop, which goes to some actuator such as a motor.

pid.*N*.enable bit in
When true, enables the PID calculations. When false, **output** is zero, and all internal integrators, etc, are reset.

pid.*N*.tune-mode bit in
When true, enables auto tune mode. When false, normal PID calculations are performed.

pid.*N*.tune-start bit io
When set to true, starts auto tuning. Cleared when the auto tuning completes.

PARAMETERS

pid.*N*.Pgain float rw
Proportional gain. Results in a contribution to the output that is the error multiplied by **Pgain**.

pid.*N*.Igain float rw
Integral gain. Results in a contribution to the output that is the integral of the error multiplied by **Igain**. For example an error of 0.02 that lasted 10 seconds would result in an integrated error (**errorI**) of 0.2, and if **Igain** is 20, the integral term would add 4.0 to the output.

pid.*N*.Dgain float rw
Derivative gain. Results in a contribution to the output that is the rate of change (derivative) of the error multiplied by **Dgain**. For example an error that changed from 0.02 to 0.03 over 0.2 seconds would result in an error derivative (**errorD**) of of 0.05, and if **Dgain** is 5, the derivative term would add 0.25 to the output.

pid.*N*.bias float rw
bias is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. **bias** is turned off when the PID loop is disabled, just like all other components of the output. If a non-zero output is needed even when the PID loop is disabled, it should be added with an external HAL sum2 block.

pid.*N*.FF0 float rw
Zero order feed-forward term. Produces a contribution to the output that is **FF0** multiplied by the commanded value. For position loops, it should usually be left at zero. For velocity loops, **FF0** can compensate for friction or motor counter-EMF and may permit better tuning if used properly.

pid.*N*.FF1 float rw
First order feed-forward term. Produces a contribution to the output that **FF1** multiplied by the derivative of the commanded value. For position loops, the contribution is proportional to speed, and can be used to compensate for friction or motor CEMF. For velocity loops, it is proportional to acceleration and can compensate for inertia. In both cases, it can result in better tuning if used properly.

pid.*N*.FF2 float rw
Second order feed-forward term. Produces a contribution to the output that is **FF2** multiplied by the second derivative of the commanded value. For position loops, the contribution is proportional to acceleration, and can be used to compensate for inertia. For velocity loops, it should usually be left at zero.

pid.*N*.deadband float rw
Defines a range of "acceptable" error. If the absolute value of **error** is less than **deadband**, it will be treated as if the error is zero. When using feedback devices such as encoders that are inherently quantized, the deadband should be set slightly more than one-half count, to prevent the control loop from hunting back and forth if the command is between two adjacent encoder values. When

the absolute value of the error is greater than the deadband, the deadband value is subtracted from the error before performing the loop calculations, to prevent a step in the transfer function at the edge of the deadband. (See **BUGS**.)

pid.*N*.maxoutput float rw
Output limit. The absolute value of the output will not be permitted to exceed **maxoutput**, unless **maxoutput** is zero. When the output is limited, the error integrator will hold instead of integrating, to prevent windup and overshoot.

pid.*N*.maxerror float rw
Limit on the internal error variable used for P, I, and D. Can be used to prevent high **Pgain** values from generating large outputs under conditions when the error is large (for example, when the command makes a step change). Not normally needed, but can be useful when tuning non-linear systems.

pid.*N*.maxerrorD float rw
Limit on the error derivative. The rate of change of error used by the **Dgain** term will be limited to this value, unless the value is zero. Can be used to limit the effect of **Dgain** and prevent large output spikes due to steps on the command and/or feedback. Not normally needed.

pid.*N*.maxerrorI float rw
Limit on error integrator. The error integrator used by the **Igain** term will be limited to this value, unless it is zero. Can be used to prevent integrator windup and the resulting overshoot during/after sustained errors. Not normally needed.

pid.*N*.maxcmdD float rw
Limit on command derivative. The command derivative used by **FF1** will be limited to this value, unless the value is zero. Can be used to prevent **FF1** from producing large output spikes if there is a step change on the command. Not normally needed.

pid.*N*.maxcmdDD float rw
Limit on command second derivative. The command second derivative used by **FF2** will be limited to this value, unless the value is zero. Can be used to prevent **FF2** from producing large output spikes if there is a step change on the command. Not normally needed.

pid.*N*.tune-type u32 rw
When set to 0, **Pgain/Igain/Dgain** are caclulated. When set to 1, **Pgain/Igain/FF1** are calculated.

pid.*N*.tune-cycles u32 rw
Determines the number of cycles to run to characterize the process. **tune-cycles** actually sets the number of half cycles. More cycles results in a more accurate characterization as the average of all cycles is used.

pid.*N*.tune-effort float rw
Determines the effor used in setting up the limit cycle in the process. **tune-effort** should be set to a positive value less than **maxoutput**. Start with something small and work up to a value that results in a good portion of the maximum motor current being used. The smaller the value, the smaller the amplitude of the limit cycle.

pid.*N*.errorI float ro (only if debug=1)
Integral of error. This is the value that is multiplied by **Igain** to produce the Integral term of the output.

pid.*N*.errorD float ro (only if debug=1)
Derivative of error. This is the value that is multiplied by **Dgain** to produce the Derivative term of the output.

pid.*N*.commandD float ro (only if debug=1)
Derivative of command. This is the value that is multiplied by **FF1** to produce the first order feedforward term of the output.

pid.*N*.commandDD float ro (only if debug=1)

Second derivative of command. This is the value that is multiplied by **FF2** to produce the second order feed-forward term of the output.

pid.*N*.ultimate-gain float ro (only if debug=1)

Determined from process characterization. **ultimate-gain** is the ratio of **tune-effort** to the limit cycle amplitude multipled by 4.0 divided by Pi. **pid.*N*.ultimate-period** float ro (only if debug=1) Determined from process characterization. **ultimate-period** is the period of the limit cycle.

BUGS

Some people would argue that deadband should be implemented such that error is treated as zero if it is within the deadband, and be unmodified if it is outside the deadband. This was not done because it would cause a step in the transfer function equal to the size of the deadband. People who prefer that behavior are welcome to add a parameter that will change the behavior, or to write their own version of **at_pid**. However, the default behavior should not be changed.

NAME

bin2gray − convert a number to the gray-code representation

SYNOPSIS

loadrt bin2gray [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

Converts a number into gray-code

FUNCTIONS

bin2gray.*N*

PINS

bin2gray.*N***.in** u32 in
binary code in

bin2gray.*N***.out** u32 out
gray code out

AUTHOR

andy pugh

LICENSE

GPL

NAME

biquad − Biquad IIR filter

SYNOPSIS

loadrt biquad [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

Biquad IIR filter. Implements the following transfer function: H(z) = (n0 + n1z-1 + n2z-2) / (1+ d1z-1 + d2z-2)

FUNCTIONS

biquad.*N* (requires a floating-point thread)

PINS

biquad.*N***.in** float in
Filter input.

biquad.*N***.out** float out
Filter output.

biquad.*N***.enable** bit in (default: *0*)
Filter enable. When false, the in is passed to out without any filtering. A transition from false to true causes filter coefficients to be calculated according to parameters

biquad.*N***.valid** bit out (default: *0*)
When false, indicates an error occured when caclulating filter coefficients.

PARAMETERS

biquad.*N***.type** u32 rw (default: *0*)
Filter type determines the type of filter coefficients calculated. When 0, coefficients must be loaded directly. When 1, a low pass filter is created. When 2, a notch filter is created.

biquad.*N***.f0** float rw (default: *250.0*)
The corner frequency of the filter.

biquad.*N***.Q** float rw (default: *0.7071*)
The Q of the filter.

biquad.*N***.d1** float rw (default: *0.0*)
1st-delayed denominator coef

biquad.*N***.d2** float rw (default: *0.0*)
2nd-delayed denominator coef

biquad.*N***.n0** float rw (default: *1.0*)
non-delayed numerator coef

biquad.*N***.n1** float rw (default: *0.0*)
1st-delayed numerator coef

biquad.*N***.n2** float rw (default: *0.0*)
2nd-delayed numerator coef

biquad.*N***.s1** float rw (default: *0.0*)
biquad.*N***.s2** float rw (default: *0.0*)

LICENSE

GPL

NAME

bitslice − Converts an unsigned-32 input into individual bits

SYNOPSIS

loadrt bitslice [count=*N***|names=***name1***[,***name2***...]] [personality=***P,P,...***]**

DESCRIPTION

This component creates individual bit-outputs for each bit of an unsigned-32 input. The number of bits can be limited by the "personality" modparam. The inverse process can be perfomed by the weighted_sum HAL component.

FUNCTIONS

bitslice.*N*

PINS

bitslice.*N***.in** u32 in

The input value

bitslice.*N***.out-***MM* bit out (MM=00..personality)

AUTHOR

Andy Pugh

LICENSE

GPL2+

NAME

bitwise − Computes various bitwise operations on the two input values

SYNOPSIS

loadrt bitwise [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

bitwise.*N*

PINS

bitwise.*N***.in0** u32 in
First input value

bitwise.*N***.in1** u32 in
Second input value

bitwise.*N***.out-and** u32 out
The bitwise AND of the two inputs

bitwise.*N***.out-or** u32 out
The bitwise OR of the two inputs

bitwise.*N***.out-xor** u32 out
The bitwise XOR of the two inputs

bitwise.*N***.out-nand** u32 out
The inverse of the bitwise AND

bitwise.*N***.out-nor** u32 out
The inverse of the bitwise OR

bitwise.*N***.out-xnor** u32 out
The inverse of the bitwise XOR

AUTHOR

Andy Pugh

LICENSE

GPL 2+

NAME

bldc − BLDC and AC-servo control component

SYNOPSIS

loadrt bldc personality=*P*

DESCRIPTION

This component is designed as an interface between the most common forms of three-phase motor feedback devices and the corresponding types of drive. However there is no requirement that the motor and drive should necessarily be of inherently compatible types.

SYNOPSIS

(ignore the auto-generated SYNOPSIS above)

loadrt bldc cfg=qi6,aH

Each instance of the component is defined by a group of letters describing the input and output types. A comma separates individual instances of the component.

Tags

Input type definitions are all lower-case.

n No motor feedback. This mode could be used to drive AC induction motors, but is also potentially useful for creating free-running motor simulators for drive testing.

h Hall sensor input. Brushless DC motors (electronically commutated permanent magnet 3-phase motors) typically use a set of three Hall sensors to measure the angular position of the rotor. A lower-case **h** in the cfg string indicates that these should be used.

a Absolute encoder input. (Also possibly used by some forms of Resolver conversion hardware). The presence of this tag over-rides all other inputs. Note that the component still requires to be be connected to the **rawcounts** encoder pin to prevent loss of commutation on index-reset.

q Incremental (quadrature) encoder input. If this input is used then the rotor will need to be homed before the motor can be run.

i Use the index of an incremental encoder as a home reference.

f Use a 4-bit Gray-scale patttern to determine rotor alignment. This scheme is only used on the Fanuc "Red Cap" motors. This mode could be used to control one of these motors using a non-Fanuc drive.

Output type descriptions are all upper-case.

Defaults The component will always calculate rotor angle, phase angle and the absolute value of the input **value** for interfacing with drives such as the Mesa 8i20. It will also default to three individual, bipolar phase output values if no other output type modifiers are used.

B Bit level outputs. Either 3 or 6 logic-level outputs indicating which high or low gate drivers on an external drive should be used.

6 Create 6 rather than the default 3 outputs. In the case of numeric value outputs these are separate positive and negative drive amplitudes. Both have positive magnitude.

H Emulated Hall sensor output. This mode can be used to control a drive which expects 3x Hall signals, or to convert between a motor with one hall pattern and a drive which expects a different one.

F Emulated Fanuc Red Cap Gray-code encoder output. This mode might be used to drive a non-Fanuc

motor using a Fanuc drive intended for the "Red-Cap" motors.

T Force Trapezoidal mode.

OPERATING MODES

The component can control a drive in either Trapezoidal or Sinusoidal mode, but will always default to sinusoidal if the input and output modes allow it. This can be over-ridden by the **T** tag. Sinusoidal commutation is significantly smoother (trapezoidal commutation induces 13% torque ripple).

ROTOR HOMING.

To use an encoder for commutation a reference 0-degrees point must be found. The component uses the convention that motor zero is the point that an unloaded motor aligns to with a positive voltage on the A (or U) terminal and the B & C (or V and W) terminals connected together and to -ve voltage. There will be two such positions on a 4-pole motor, 3 on a 6-pole and so on. They are all functionally equivalent as far as driving the motor is concerned. If the motor has Hall sensors then the motor can be started in trapezoidal commutation mode, and will switch to sinusoidal commutation when an alignment is found. If the mode is **qh** then the first Hall state-transition will be used. If the mode is **qhi** then the encoder index will be used. This gives a more accurate homing position if the distance in encoder counts between motor zero and encoder index is known. To force homing to the Hall edges instead simply omit the **i**.

Motors without Hall sensors may be homed in synchronous/direct mode. The better of these options is to home to the encoder zero using the **iq** config parameter. When the **init** pin goes high the motor will rotate (in a direction determined by the **rev** pin) until the encoder indicates an index-latch (the servo thread runs too slowly to rely on detecting an encoder index directly). If there is no encoder index or its location relative to motor zero can not be found, then an alternative is to use *magnetic* homing using the **q** config. In this mode the motor will go through an alignment sequence ending at motor zero when the init pin goes high It will then set the final position as motor zero. Unfortunately the motor is rather *springy* in this mode and so alignment is likely to be fairly sensitive to load.

FUNCTIONS

bldc.*N* (requires a floating-point thread)

PINS

bldc.*N***.hall1** bit in [if personality & 0x01]
Hall sensor signal 1

bldc.*N***.hall2** bit in [if personality & 0x01]
Hall sensor signal 2

bldc.*N***.hall3** bit in [if personality & 0x01]
Hall sensor signal 3

bldc.*N***.hall-error** bit out [if personality & 0x01]
Indicates that the selected hall pattern gives inconsistent rotor position data. This can be due to the pattern being wrong for the motor, or one or more sensors being unconnected or broken. A consistent pattern is not neceesarily valid, but an inconsistent one can never be valid.

bldc.*N***.C1** bit in [if (personality & 0x10)]
Fanuc Gray-code bit 0 input

bldc.*N***.C2** bit in [if (personality & 0x10)]
Fanuc Gray-code bit 1 input

bldc.*N***.C4** bit in [if (personality & 0x10)]
Fanuc Gray-code bit 2 input

bldc.*N*.C8 bit in [if (personality & 0x10)]
Fanuc Gray-code bit 3 input

bldc.*N*.value float in
PWM master amplitude input

bldc.*N*.lead-angle float in [if personality & 0x06] (default: *90*)
The phase lead between the electrical vector and the rotor position in degrees

bldc.*N*.rev bit in
Set this pin true to reverse the motor. Negative PWM amplitudes will also reverse the motor and there will generally be a Hall pattern that runs the motor in each direction too.

bldc.*N*.frequency float in [if (personality & 0x0F) == 0]
Frequency input for motors with no feedback at all, or those with only an index (which is ignored)

bldc.*N*.initvalue float in [if personality & 0x04] (default: *0.2*)
The current to be used for the homing sequence in applications where an incremental encoder is used with no hall-sensor feedback

bldc.*N*.rawcounts s32 in [if personality & 0x06] (default: *0*)
Encoder counts input. This must be linked to the encoder rawcounts pin or encoder index resets will cause the motor commutation to fail

bldc.*N*.index-enable bit io [if personality & 0x08]
This pin should be connected to the associated encoder index-enable pin to zero the encoder when it passes index This is only used indicate to the bldc control component that an index has been seen

bldc.*N*.init bit in [if (personality & 0x05) == 4]
A rising edge on this pin starts the motor alignment sequence. This pin should be connected in such a way that the motors re-align any time that encoder monitoring has been interrupted. Typically this will only be at machine power-off. The alignment process involves powering the motor phases in such a way as to put the motor in a known position. The encoder counts are then stored in the **offset** parameter. The alignement process will tend to cause a following error if it is triggered while the axis is enabled, so should be set before the matching axis.N.enable pin. The complementary **init-done** pin can be used to handle the required sequencing.

Both pins can be ignored if the encoder offset is known explicitly, such as is the case with an absolute encoder. In that case the **offset** parameter can be set directly in the HAL file

bldc.*N*.init-done bit out [if (personality & 0x05) == 4] (default: *0*)
Indicates homing sequence complete

bldc.*N*.A-value float out [if (personality & 0xF00) == 0]
Output amplitude for phase A

bldc.*N*.B-value float out [if (personality & 0xF00) == 0]
Output amplitude for phase B

bldc.*N*.C-value float out [if (personality & 0xF00) == 0]
Output amplitude for phase C

bldc.*N*.A-on bit out [if (personality & 0xF00) == 0x100]
Output bit for phase A

bldc.*N*.B-on bit out [if (personality & 0xF00) == 0x100]
Output bit for phase B

bldc.*N*.C-on bit out [if (personality & 0xF00) == 0x100]
Output bit for phase C

bldc.*N*.A-high float out [if (personality & 0xF00) == 0x200]
High-side driver for phase A

bldc.*N*.B-high float out [if (personality & 0xF00) == 0x200]
High-side driver for phase B

bldc.*N*.C-high float out [if (personality & 0xF00) == 0x200]
High-side driver for phase C

bldc.*N*.A-low float out [if (personality & 0xF00) == 0x200]
Low-side driver for phase A

bldc.*N*.B-low float out [if (personality & 0xF00) == 0x200]
Low-side driver for phase B

bldc.*N*.C-low float out [if (personality & 0xF00) == 0x200]
Low-side driver for phase C

bldc.*N*.A-high-on bit out [if (personality & 0xF00) == 0x300]
High-side driver for phase A

bldc.*N*.B-high-on bit out [if (personality & 0xF00) == 0x300]
High-side driver for phase B

bldc.*N*.C-high-on bit out [if (personality & 0xF00) == 0x300]
High-side driver for phase C

bldc.*N*.A-low-on bit out [if (personality & 0xF00) == 0x300]
Low-side driver for phase A

bldc.*N*.B-low-on bit out [if (personality & 0xF00) == 0x300]
Low-side driver for phase B

bldc.*N*.C-low-on bit out [if (personality & 0xF00) == 0x300]
Low-side driver for phase C

bldc.*N*.hall1-out bit out [if (personality & 0x400)]
Hall 1 output

bldc.*N*.hall2-out bit out [if (personality & 0x400)]
Hall 2 output

bldc.*N*.hall3-out bit out [if (personality & 0x400)]
Hall 3 output

bldc.*N*.C1-out bit out [if (personality & 0x800)]
Fanuc Gray-code bit 0 output

bldc.*N*.C2-out bit out [if (personality & 0x800)]
Fanuc Gray-code bit 1 output

bldc.*N*.C4-out bit out [if (personality & 0x800)]
Fanuc Gray-code bit 2 output

bldc.*N*.C8-out bit out [if (personality & 0x800)]
Fanuc Gray-code bit 3 output

bldc.*N*.phase-angle float out (default: *0*)
Phase angle including lead/lag angle after encoder zeroing etc. Useful for angle/current drives. This value has a range of 0 to 1 and measures electrical revolutions. It will have two zeros for a 4 pole motor, three for a 6-pole etc

bldc.*N*.rotor-angle float out (default: *0*)
Rotor angle after encoder zeroing etc. Useful for angle/current drives which add their own phase offset such as the 8i20. This value has a range of 0 to 1 and measures electrical revolutions. It will have two zeros for a 4 pole motor, three for a 6-pole etc

bldc.*N*.out float out
Current output, including the effect of the dir pin and the alignment sequence

bldc.*N*.out-dir bit out
Direction output, high if /fBvalue/fR is negative XOR /fBrev/fR is true.

bldc.*N*.out-abs float out
Absolute value of the input value

PARAMETERS

bldc.*N*.in-type s32 r (default: *-1*)
state machine output, will probably hide after debug

bldc.*N*.out-type s32 r (default: *-1*)
state machine output, will probably hide after debug

bldc.*N*.scale s32 rw [if personality & 0x06] (default: *512*)
The number of encoder counts per rotor revolution.

bldc.*N*.poles s32 rw [if personality & 0x06] (default: *4*)
The number of motor poles. The encoder scale will be divided by this value to determine the number of encoder counts per electrical revolution

bldc.*N*.encoder-offset s32 rw [if personality & 0x0A] (default: *0*)
The offset, in encoder counts, between the motor electrical zero and the encoder zero modulo the number of counts per electrical revolution

bldc.*N*.offset-measured s32 r [if personality & 0x04] (default: *0*)
The encoder offset measured by the homing sequence (in certain modes)

bldc.*N*.drive-offset float rw (default: *0*)
The angle, in degrees, applied to the commanded angle by the drive in degrees. This value is only used during the homing sequence of drives with incremental encoder feedback. It is used to back-calculate from commanded angle to actual phase angle. It is only relevant to drives which expect rotor-angle input rather than phase-angle demand. Should be 0 for most drives.

bldc.*N*.output-pattern u32 rw [if personality & 0x400] (default: *25*)
Commutation pattern to be output in Hall Signal translation mode. See the description of /fBpattern/fR for details

bldc.*N*.pattern u32 rw [if personality & 0x01] (default: *25*)
Commutation pattern to use, from 0 to 47. Default is type 25. Every plausible combination is included. The table shows the excitation pattern along the top, and the pattern code on the left hand side. The table entries are the hall patterns in H1, H2, H3 order. Common patterns are: 0 (30 degree commutation) and 26, its reverse. 17 (120 degree). 18 (alternate 60 degree). 21 (300 degree, Bodine). 22 (240 degree). 25 (60 degree commutation).

Note that a number of incorrect commutations will have non-zero net torque which might look as if they work, but don't really.

If your motor lacks documentation it might be worth trying every pattern.

Phases, Source - Sink						
pat	**B-A**	**C-A**	**C-B**	**A-B**	**A-C**	**B-C**
0	000	001	011	111	110	100
1	001	000	010	110	111	101
2	000	010	011	111	101	100
3	001	011	010	110	100	101
4	010	011	001	101	100	110
5	011	010	000	100	101	111
6	010	000	001	101	111	110
7	011	001	000	100	110	111
8	000	001	101	111	110	010
9	001	000	100	110	111	011
10	000	010	110	111	101	001
11	001	011	111	110	100	000
12	010	011	111	101	100	000
13	011	010	110	100	101	001
14	010	000	100	101	111	011
15	011	001	101	100	110	010
16	000	100	101	111	011	010
17	001	101	100	110	010	011
18	000	100	110	111	011	001
19	001	101	111	110	010	000
20	010	110	111	101	001	000
21	011	111	110	100	000	001
22	010	110	100	101	001	011
23	011	111	101	100	000	010
24	100	101	111	011	010	000
25	101	100	110	010	011	001
26	100	110	111	011	001	000
27	101	111	110	010	000	001
28	110	111	101	001	000	010
29	111	110	100	000	001	011
30	110	100	101	001	011	010
31	111	101	100	000	010	011
32	100	101	001	011	010	110
33	101	100	000	010	011	111
34	100	110	010	011	001	101
35	101	111	011	010	000	100
36	110	111	011	001	000	100
37	111	110	010	000	001	101
38	110	100	000	001	011	111
39	111	101	001	000	010	110
40	100	000	001	011	111	110
41	101	001	000	010	110	111
42	100	000	010	011	111	101
43	101	001	011	010	110	100
44	110	010	011	001	101	100
45	111	011	010	000	100	101
46	110	010	000	001	101	111
47	111	011	001	000	100	110

AUTHOR

Andy Pugh

LICENSE

GPL

NAME

bldc_hall3 − 3-wire BLDC motor driver using Hall sensors and trapezoidal commutation.

SYNOPSIS

The functionality of this component is now included in the generic "bldc" component. This component is likely to be removed in a future release

DESCRIPTION

This component produces a 3-wire bipolar output. This suits upstream drivers that interpret a negative input as a low-side drive and positive as a high-side drive. This includes the Hostmot2 3pwmgen function, which is likely to be the most common application of this component.

FUNCTIONS

bldc-hall3.*N* (requires a floating-point thread)
Interpret Hall sensor patterns and set 3-phase amplitudes

PINS

bldc-hall3.*N***.hall1** bit in
Hall sensor signal 1

bldc-hall3.*N***.hall2** bit in
Hall sensor signal 2

bldc-hall3.*N***.hall3** bit in
Hall sensor signal 3

bldc-hall3.*N***.value** float in
PWM master amplitude input

bldc-hall3.*N***.dir** bit in
Forwards / reverse selection. Negative PWM amplitudes will also reverse the motor and there will generally be a pattern that runs the motor in each direction too.

bldc-hall3.*N***.A-value** float out
Output amplitude for phase A

bldc-hall3.*N***.B-value** float out
Output amplitude for phase B

bldc-hall3.*N***.C-value** float out
Output amplitude for phase C

PARAMETERS

bldc-hall3.*N***.pattern** u32 rw (default: *25*)
Commutation pattern to use, from 0 to 47. Default is type 25. Every plausible combination is included. The table shows the excitation pattern along the top, and the pattern code on the left hand side. The table entries are the hall patterns in H1, H2, H3 order. Common patterns are: 0 (30 degree commutation) and 26, its reverse. 17 (120 degree). 18 (alternate 60 degree). 21 (300 degree, Bodine). 22 (240 degree). 25 (60 degree commutation).

Note that a number of incorrect commutations will have non-zero net torque which might look as if they work, but don’t really.

If your motor lacks documentation it might be worth trying every pattern.

Phases, Source - Sink						
pat	**B-A**	**C-A**	**C-B**	**A-B**	**A-C**	**B-C**
0	000	001	011	111	110	100
1	001	000	010	110	111	101
2	000	010	011	111	101	100
3	001	011	010	110	100	101
4	010	011	001	101	100	110
5	011	010	000	100	101	111
6	010	000	001	101	111	110
7	011	001	000	100	110	111
8	000	001	101	111	110	010
9	001	000	100	110	111	011
10	000	010	110	111	101	001
11	001	011	111	110	100	000
12	010	011	111	101	100	000
13	011	010	110	100	101	001
14	010	000	100	101	111	011
15	011	001	101	100	110	010
16	000	100	101	111	011	010
17	001	101	100	110	010	011
18	000	100	110	111	011	001
19	001	101	111	110	010	000
20	010	110	111	101	001	000
21	011	111	110	100	000	001
22	010	110	100	101	001	011
23	011	111	101	100	000	010
24	100	101	111	011	010	000
25	101	100	110	010	011	001
26	100	110	111	011	001	000
27	101	111	110	010	000	001
28	110	111	101	001	000	010
29	111	110	100	000	001	011
30	110	100	101	001	011	010
31	111	101	100	000	010	011
32	100	101	001	011	010	110
33	101	100	000	010	011	111
34	100	110	010	011	001	101
35	101	111	011	010	000	100
36	110	111	011	001	000	100
37	111	110	010	000	001	101
38	110	100	000	001	011	111
39	111	101	001	000	010	110
40	100	000	001	011	111	110
41	101	001	000	010	110	111
42	100	000	010	011	111	101
43	101	001	011	010	110	100
44	110	010	011	001	101	100
45	111	011	010	000	100	101
46	110	010	000	001	101	111
47	111	011	001	000	100	110

SEE ALSO

bldc_hall6 6-wire unipolar driver for BLDC motors.

AUTHOR

Andy Pugh

LICENSE

GPL

NAME

blend − Perform linear interpolation between two values

SYNOPSIS

loadrt blend [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

blend.*N* (requires a floating-point thread)

PINS

blend.*N***.in1** float in
First input. If select is equal to 1.0, the output is equal to in1

blend.*N***.in2** float in
Second input. If select is equal to 0.0, the output is equal to in2

blend.*N***.select** float in
Select input. For values between 0.0 and 1.0, the output changes linearly from in2 to in1

blend.*N***.out** float out
Output value.

PARAMETERS

blend.*N***.open** bit rw
If true, select values outside the range 0.0 to 1.0 give values outside the range in2 to in1. If false, outputs are clamped to the the range in2 to in1

LICENSE

GPL

NAME

charge_pump − Create a square-wave for the ’charge pump’ input of some controller boards

SYNOPSIS

loadrt charge_pump

DESCRIPTION

The ’Charge Pump’ should be added to the base thread function. When enabled the output is on for one period and off for one period. To calculate the frequency of the output 1/(period time in seconds x 2) = hz. For example if you have a base period of 100,000ns that is 0.0001 seconds and the formula would be 1/(0.0001 x 2) = 5,000 hz or 5 Khz

FUNCTIONS

charge-pump

Toggle the output bit (if enabled)

PINS

charge-pump.out bit out

Square wave if ’enable’ is TRUE or unconnected, low if ’enable’ is FALSE

charge-pump.enable bit in (default: *TRUE*)

If FALSE, forces ’out’ to be low

LICENSE

GPL

NAME

clarke2 − Two input version of Clarke transform

SYNOPSIS

loadrt clarke2 [count=*N***|names=***name1***[,***name2...***]]**

DESCRIPTION

The Clarke transform can be used to translate a vector quantity from a three phase system (three components 120 degrees apart) to a two phase Cartesian system.

clarke2 implements a special case of the Clarke transform, which only needs two of the three input phases. In a three wire three phase system, the sum of the three phase currents or voltages must always be zero. As a result only two of the three are needed to completely define the current or voltage. **clarke2** assumes that the sum is zero, so it only uses phases A and B of the input. Since the H (homopolar) output will always be zero in this case, it is not generated.

FUNCTIONS

clarke2.*N* (requires a floating-point thread)

PINS

clarke2.*N***.a** float in
clarke2.*N***.b** float in
first two phases of three phase input

clarke2.*N***.x** float out
clarke2.*N***.y** float out
cartesian components of output

SEE ALSO

clarke3 for the general case, **clarkeinv** for the inverse transform.

LICENSE

GPL

NAME

clarke3 − Clarke (3 phase to cartesian) transform

SYNOPSIS

loadrt clarke3 [count=*N***|names=***name1***[,***name2...***]]**

DESCRIPTION

The Clarke transform can be used to translate a vector quantity from a three phase system (three components 120 degrees apart) to a two phase Cartesian system (plus a homopolar component if the three phases don't sum to zero).

clarke3 implements the general case of the transform, using all three phases. If the three phases are known to sum to zero, see **clarke2** for a simpler version.

FUNCTIONS

clarke3.*N* (requires a floating-point thread)

PINS

clarke3.*N***.a** float in
clarke3.*N***.b** float in
clarke3.*N***.c** float in
three phase input vector

clarke3.*N***.x** float out
clarke3.*N***.y** float out
cartesian components of output

clarke3.*N***.h** float out
homopolar component of output

SEE ALSO

clarke2 for the 'a+b+c=0' case, **clarkeinv** for the inverse transform.

LICENSE

GPL

NAME

clarkeinv − Inverse Clarke transform

SYNOPSIS

loadrt clarkeinv [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

The inverse Clarke transform can be used rotate a vector quantity and then translate it from Cartesian coordinate system to a three phase system (three components 120 degrees apart).

FUNCTIONS

clarkeinv.*N* (requires a floating-point thread)

PINS

clarkeinv.*N***.x** float in
clarkeinv.*N***.y** float in
cartesian components of input

clarkeinv.*N***.h** float in
homopolar component of input (usually zero)

clarkeinv.*N***.theta** float in
rotation angle: 0.00 to 1.00 = 0 to 360 degrees

clarkeinv.*N***.a** float out
clarkeinv.*N***.b** float out
clarkeinv.*N***.c** float out
three phase output vector

SEE ALSO

clarke2 and **clarke3** for the forward transform.

LICENSE

GPL

NAME

classicladder − realtime software plc based on ladder logic

SYNOPSIS

loadrt classicladder_rt [numRungs=*N***] [numBits=***N***] [numWords=***N***] [numTimers=***N***] [numMonostables=***N***] [numCounters=***N***] [numPhysInputs=***N***] [numPhysOutputs=***N***] [numArithmExpr=***N***] [numSections=***N***] [numSymbols=***N***] [numS32in=***N***] [numS32out=***N***] [numFloatIn=***N***] [numFloatOut=***N***]**

DESCRIPTION

These pins and parameters are created by the realtime **classicladder_rt** module. Each period (minimum 1000000 ns), classicladder reads the inputs, evaluates the ladder logic defined in the GUI, and then writes the outputs.

PINS

classicladder.0.in-*NN* IN bit

These bit signal pins map to **%I***NN* variables in classicladder

classicladder.0.out-*NN* OUT bit

These bit signal pins map to **%Q***NN* variables in classicladder Output from classicladder

classicladder.0.s32in-*NN* IN s32

Integer input from classicladder These s32 signal pins map to **%IW***NN* variables in classicladder

classicladder.0.s32out-*NN* OUT s32

Integer output from classicladder These s32 signal pins map to **%QW***NN* variables in classicladder

classicladder.0.floatin-*NN* IN float

Integer input from classicladder These float signal pins map to **%IF***NN* variables in classicladder These are truncated to S32 values internally. eg 7.5 will be 7

classicladder.0.floatout-*NN* OUT float

Float output from classicladder These float signal pins map to **%QF***NN* variables in classicladder

PARAMETERS

classicladder.0.refresh.time RO s32

Tells you how long the last refresh took

classicladder.0.refresh.tmax RW s32

Tells you how long the longest refresh took

classicladder.0.ladder-state RO s32

Tells you if the program is running or not

FUNCTIONS

classicladder.0.refresh FP

The rung update rate. Add this to the servo thread. You can added it to a faster thread but it Will update no faster than once every 1 millisecond (1000000 ns).

BUGS

See http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder_Ver_7.124 for the latest.

SEE ALSO

Classicladder chapters in the LinuxCNC documentation for a full description of the **Classicladder** syntax and examples

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder_Ver_7.124

NAME

comp − Two input comparator with hysteresis

SYNOPSIS

loadrt comp [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

comp.*N* (requires a floating-point thread)
Update the comparator

PINS

comp.*N***.in0** float in
Inverting input to the comparator

comp.*N***.in1** float in
Non-inverting input to the comparator

comp.*N***.out** bit out
Normal output. True when **in1** > **in0** (see parameter **hyst** for details)

comp.*N***.equal** bit out
Match output. True when difference between **in1** and **in0** is less than **hyst**/2

PARAMETERS

comp.*N***.hyst** float rw (default: *0.0*)
Hysteresis of the comparator (default 0.0)

With zero hysteresis, the output is true when **in1** > **in0**. With nonzero hysteresis, the output switches on and off at two different values, separated by distance **hyst** around the point where **in1** = **in0**. Keep in mind that floating point calculations are never absolute and it is wise to always set **hyst** if you intend to use equal

LICENSE

GPL

NAME

constant − Use a parameter to set the value of a pin

SYNOPSIS

loadrt constant [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

constant.*N* (requires a floating-point thread)

PINS

constant.*N***.out** float out

PARAMETERS

constant.*N***.value** float rw

LICENSE

GPL

NAME

conv_bit_s32 − Convert a value from bit to s32

SYNOPSIS

loadrt conv_bit_s32 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-bit-s32.*N*

Update ’out’ based on ’in’

PINS

conv-bit-s32.*N***.in** bit in
conv-bit-s32.*N***.out** s32 out

LICENSE

GPL

NAME

conv_bit_u32 − Convert a value from bit to u32

SYNOPSIS

loadrt conv_bit_u32 [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

conv-bit-u32.*N*

Update ’out’ based on ’in’

PINS

conv-bit-u32.*N***.in** bit in
conv-bit-u32.*N***.out** u32 out

LICENSE

GPL

NAME

conv_float_s32 − Convert a value from float to s32

SYNOPSIS

loadrt conv_float_s32 [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

conv-float-s32.*N* (requires a floating-point thread)

Update ’out’ based on ’in’

PINS

conv-float-s32.*N***.in** float in

conv-float-s32.*N***.out** s32 out

conv-float-s32.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of s32

PARAMETERS

conv-float-s32.*N***.clamp** bit rw

If TRUE, then clamp to the range of s32. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

conv_float_u32 − Convert a value from float to u32

SYNOPSIS

loadrt conv_float_u32 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-float-u32.*N* (requires a floating-point thread)

Update ’out’ based on ’in’

PINS

conv-float-u32.*N***.in** float in

conv-float-u32.*N***.out** u32 out

conv-float-u32.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of u32

PARAMETERS

conv-float-u32.*N***.clamp** bit rw

If TRUE, then clamp to the range of u32. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

conv_s32_bit − Convert a value from s32 to bit

SYNOPSIS

loadrt conv_s32_bit [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-s32-bit.*N*

Update ’out’ based on ’in’

PINS

conv-s32-bit.*N***.in** s32 in

conv-s32-bit.*N***.out** bit out

conv-s32-bit.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of bit

PARAMETERS

conv-s32-bit.*N***.clamp** bit rw

If TRUE, then clamp to the range of bit. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

conv_s32_float − Convert a value from s32 to float

SYNOPSIS

loadrt conv_s32_float [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

conv-s32-float.*N* (requires a floating-point thread)

Update ’out’ based on ’in’

PINS

conv-s32-float.*N***.in** s32 in

conv-s32-float.*N***.out** float out

LICENSE

GPL

NAME

conv_s32_u32 − Convert a value from s32 to u32

SYNOPSIS

loadrt conv_s32_u32 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-s32-u32.*N*

Update ’out’ based on ’in’

PINS

conv-s32-u32.*N***.in** s32 in

conv-s32-u32.*N***.out** u32 out

conv-s32-u32.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of u32

PARAMETERS

conv-s32-u32.*N***.clamp** bit rw

If TRUE, then clamp to the range of u32. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

conv_u32_bit − Convert a value from u32 to bit

SYNOPSIS

loadrt conv_u32_bit [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

conv-u32-bit.*N*

Update ’out’ based on ’in’

PINS

conv-u32-bit.*N***.in** u32 in

conv-u32-bit.*N***.out** bit out

conv-u32-bit.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of bit

PARAMETERS

conv-u32-bit.*N***.clamp** bit rw

If TRUE, then clamp to the range of bit. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

conv_u32_float − Convert a value from u32 to float

SYNOPSIS

loadrt conv_u32_float [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-u32-float.*N* (requires a floating-point thread)

Update ’out’ based on ’in’

PINS

conv-u32-float.*N***.in** u32 in

conv-u32-float.*N***.out** float out

LICENSE

GPL

NAME

conv_u32_s32 − Convert a value from u32 to s32

SYNOPSIS

loadrt conv_u32_s32 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

conv-u32-s32.*N*

Update ’out’ based on ’in’

PINS

conv-u32-s32.*N***.in** u32 in

conv-u32-s32.*N***.out** s32 out

conv-u32-s32.*N***.out-of-range** bit out

TRUE when ’in’ is not in the range of s32

PARAMETERS

conv-u32-s32.*N***.clamp** bit rw

If TRUE, then clamp to the range of s32. If FALSE, then allow the value to "wrap around".

LICENSE

GPL

NAME

counter − counts input pulses **(DEPRECATED)**

SYNOPSIS

loadrt counter [num_chan=*N*]

DESCRIPTION

counter is a deprecated HAL component and will be removed in a future release. Use the **encoder** component with encoder.X.counter-mode set to TRUE.

counter is a HAL component that provides software- based counting that is useful for spindle position sensing and maybe other things. Instead of using a real encoder that outputs quadrature, some lathes have a sensor that generates a simple pulse stream as the spindle turns and an index pulse once per revolution. This component simply counts up when a "count" pulse (phase-A) is received, and if reset is enabled, resets when the "index" (phase-Z) pulse is received.

This is of course only useful for a unidirectional spindle, as it is not possible to sense the direction of rotation.

counter conforms to the "canonical encoder" interface described in the HAL manual.

FUNCTIONS

counter.capture-position (uses floating-point)
Updates the counts, position and velocity outputs based on internal counters.

counter.update-counters
Samples the phase-A and phase-Z inputs and updates internal counters.

PINS

counter.*N*.phase-A bit in
The primary input signal. The internal counter is incremented on each rising edge.

counter.*N*.phase-Z bit in
The index input signal. When the **index-enable** pin is TRUE and a rising edge on **phase-Z** is seen, **index-enable** is set to FALSE and the internal counter is reset to zero.

counter.*N*.index-enable bit io
counter.*N*.reset bit io
counter.*N*.counts signed out
counter.*N*.position float out
counter.*N*.velocity float out
These pins function according to the canonical digital encoder interface.

counter.*N*.position-scale float rw
This parameter functions according to the canonical digital encoder interface.

counter.*N*.rawcounts signed ro
The internal counts value, updated from **update-counters** and reflected in the output pins at the next call to **capture-position**.

SEE ALSO

encoder(9). in the LinuxCNC documentation.

NAME

ddt − Compute the derivative of the input function

SYNOPSIS

loadrt ddt [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

ddt.*N* (requires a floating-point thread)

PINS

ddt.*N***.in** float in
ddt.*N***.out** float out

LICENSE

GPL

NAME

deadzone − Return the center if within the threshold

SYNOPSIS

loadrt deadzone [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

deadzone.*N* (requires a floating-point thread)
Update **out** based on **in** and the parameters.

PINS

deadzone.*N***.in** float in
deadzone.*N***.out** float out

PARAMETERS

deadzone.*N***.center** float rw (default: *0.0*)
The center of the dead zone

deadzone.*N***.threshhold** float rw (default: *1.0*)
The dead zone is **center** ± (**threshhold**/2)

LICENSE

GPL

NAME

debounce − filter noisy digital inputs

SYNOPSIS

loadrt debounce cfg=*size[,size,...]*

Creates debounce groups with the number of filters specified by (*size*). Every filter in the same group has the same sample rate and delay.

DESCRIPTION

The debounce filter works by incrementing a counter whenever the input is true, and decrementing the counter when it is false. If the counter decrements to zero, the output is set false and the counter ignores further decrements. If the counter increments up to a threshold, the output is set true and the counter ignores further increments. If the counter is between zero and the threshold, the output retains its previous state. The threshold determines the amount of filtering: a threshold of 1 does no filtering at all, and a threshold of N requires a signal to be present for N samples before the output changes state.

FUNCTIONS

debounce.*G*

Sample all the input pins in group G and update the output pins.

PINS

debounce.*G***.***F***.in** bit in

The F’th input pin in group G.

debounce.*G***.***F***.out** bit out

The F’th output pin in group G. Reflects the last "stable" input seen on the corresponding input pin.

debounce.*G***.delay** signed rw

Sets the amount of filtering for all pins in group G.

NAME

edge − Edge detector

SYNOPSIS

loadrt edge [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

edge.*N* Produce output pulses from input edges

PINS

edge.*N***.in** bit in

edge.*N***.out** bit out

Goes high when the desired edge is seen on ’in’

edge.*N***.out-invert** bit out

Goes low when the desired edge is seen on ’in’

PARAMETERS

edge.*N***.in-edge** bit rw (default: *TRUE*)

Selects the desired edge: TRUE means falling, FALSE means rising

edge.*N***.out-width-ns** s32 rw (default: *0*)

Time in nanoseconds of the output pulse

edge.*N***.time-left-ns** s32 r

Time left in this output pulse

edge.*N***.last-in** bit r

Previous input value

LICENSE

GPL

NAME

encoder − software counting of quadrature encoder signals

SYNOPSIS

loadrt encoder [num_chan=*num* **| names=***name1***[,***name2...***]]**

DESCRIPTION

encoder is used to measure position by counting the pulses generated by a quadrature encoder. As a software-based implementation it is much less expensive than hardware, but has a limited maximum count rate. The limit is in the range of 10KHz to 50KHz, depending on the computer speed and other factors. If better performance is needed, a hardware encoder counter is a better choice. Some hardware-based systems can count at MHz rates.

encoder supports a maximum of eight channels. The number of channels actually loaded is set by the **num_chan** argument when the module is loaded. Alternatively, specify names= and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is three.

encoder has a one-phase, unidirectional mode called *counter*. In this mode, the **phase-B** input is ignored; the counts increase on each rising edge of **phase-A**. This mode may be useful for counting a unidirectional spindle with a single input line, though the noise-resistant characteristics of quadrature are lost.

FUNCTIONS

encoder.update-counters (no floating-point)

Does the actual counting, by sampling the encoder signals and decoding the quadrature waveforms. Must be called as frequently as possible, preferably twice as fast as the maximum desired count rate. Operates on all channels at once.

encoder.capture-position (uses floating point)

Captures the raw counts from **update-counters** and performs scaling and other necessary conversion, handles counter rollover, etc. Can (and should) be called less frequently than **update-counters**. Operates on all channels at once.

NAMING

The names for pins and parameters are prefixed as:
encoder.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **encoder.N.** format is shown in the following descriptions.

PINS

encoder.*N*.counter-mode bit i/o

Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false (the default), it counts in quadrature mode.

encoder.*N*.counts s32 out

Position in encoder counts.

encoder.*N*.index-enable bit i/o

When true, **counts** and **position** are reset to zero on the next rising edge of **Phase-Z**. At the same time, **index-enable** is reset to zero to indicate that the rising edge has occurred.

encoder.*N*.phase-A bit in
Quadrature input for encoder channel *N*.

encoder.*N*.phase-B bit in
Quadrature input.

encoder.*N*.phase-Z bit in
Index pulse input.

encoder.*N*.position float out
Position in scaled units (see **position-scale**)

encoder.*N*.position-interpolated float out
Position in scaled units, interpolated between encoder counts. Only valid when velocity is approximately constant and above **min-velocity-estimate**. Do not use for position control.

encoder.*N*.position-scale float i/o
Scale factor, in counts per length unit. For example, if **position-scale** is 500, then 1000 counts of the encoder will be reported as a position of 2.0 units.

encoder.*N*.rawcounts s32 out
The raw count, as determined by **update-counters**. This value is updated more frequently than **counts** and **position**. It is also unaffected by **reset** or the index pulse.

encoder.*N*.reset bit in
When true, **counts** and **position** are reset to zero immediately.

encoder.*N*.velocity float out
Velocity in scaled units per second. **encoder** uses an algorithm that greatly reduces quantization noise as compared to simply differentiating the **position** output. When the magnitude of the true velocity is below min-velocity-estimate, the velocity output is 0.

encoder.*N*.x4-mode bit i/o
Enables times-4 mode. When true (the default), the counter counts each edge of the quadrature waveform (four counts per full cycle). When false, it only counts once per full cycle. In **counter-mode**, this parameter is ignored.

encoder.*N*.latch-input bit in
encoder.*N*.latch-falling bit in (default: **TRUE**)
encoder.*N*.latch-rising bit in (default: **TRUE**)
encoder.*N*.counts-latched s32 out
encoder.*N*.position-latched float out
Update **counts-latched** and **position-latched** on the rising and/or falling edges of **latch-input** as indicated by **latch-rising** and **latch-falling**.

encoder.*N*.counter-mode bit rw
Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false (the default), it counts in quadrature mode. **encoder.*N*.capture-position.tmax** s32 rw Maximum number of CPU cycles it took to execute this function.

PARAMETERS

Parameter names for num_chan= specifier are:
encoder.*N*.the_parameter_name

Parameter names for names= specifier are:
nameN.the_parameter_name

encoder.*N*.min-velocity-estimate float rw (default: 1.0)
Determine the minimum true velocity magnitude at which **velocity** will be estimated as nonzero and **postition-interpolated** will be interpolated. The units of **min-velocity-estimate** are the same as the units of **velocity**. Setting this parameter too low will cause it to take a long time for **velocity**

to go to 0 after encoder pulses have stopped arriving.

NAME

encoder_ratio − an electronic gear to synchronize two axes

SYNOPSIS

loadrt encoder_ratio [num_chan=*num* **| names=***name1***[,***name2***...]]**

DESCRIPTION

encoder_ratio can be used to synchronize two axes (like an "electronic gear"). It counts encoder pulses from both axes in software, and produces an error value that can be used with a PID loop to make the slave encoder track the master encoder with a specific ratio.

This module supports up to eight axis pairs. The number of pairs is set by the module parameter **num_chan**. Alternatively, specify **names=** and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is one.

FUNCTIONS

encoder-ratio.sample

Read all input pins. Must be called at twice the maximum desired count rate.

encoder-ratio.update (uses floating-point)

Updates all output pins. May be called from a slower thread.

NAMING

The names for pins and parameters are prefixed as:
encoder-ratio.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **encoder-ratio.N.** format is shown in the following descriptions.

PINS

encoder-ratio.*N*.master-A bit in
encoder-ratio.*N*.master-B bit in
encoder-ratio.*N*.slave-A bit in
encoder-ratio.*N*.slave-B bit in

The encoder channels of the master and slave axes

encoder-ratio.*N*.enable bit in

When the enable pin is FALSE, the error pin simply reports the slave axis position, in revolutions. As such, it would normally be connected to the feedback pin of a PID block for closed loop control of the slave axis. Normally the command input of the PID block is left unconnected (zero), so the slave axis simply sits still. However when the enable input goes TRUE, the error pin becomes the slave position minus the scaled master position. The scale factor is the ratio of master teeth to slave teeth. As the master moves, error becomes non-zero, and the PID loop will drive the slave axis to track the master.

encoder-ratio.*N*.error float out

The error in the position of the slave (in revolutions)

PARAMETERS

encoder-ratio.N.master-ppr unsigned rw
encoder-ratio.N.slave-ppr unsigned rw

The number of pulses per revolution of the master and slave axes

encoder-ratio.N.master-teeth unsigned rw
encoder-ratio.N.slave-teeth unsigned rw
The number of "teeth" on the master and slave gears.

SEE ALSO

encoder(9)

NAME

estop_latch − ESTOP latch which sets ok-out true and fault-out false only if ok-in is true, fault-in is false, and a rising edge is seen on reset. While ok-out is true, watchdog toggles, and can be used for chargepumps or similar needs.

SYNOPSIS

loadrt estop_latch [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

estop-latch.*N*

PINS

estop-latch.*N***.ok-in** bit in
estop-latch.*N***.fault-in** bit in
estop-latch.*N***.reset** bit in
estop-latch.*N***.ok-out** bit out
estop-latch.*N***.fault-out** bit out
estop-latch.*N***.watchdog** bit out

LICENSE

GPL

NAME

feedcomp − Multiply the input by the ratio of current velocity to the feed rate

SYNOPSIS

loadrt feedcomp [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

feedcomp.*N* (requires a floating-point thread)

PINS

feedcomp.*N***.out** float out
Proportionate output value

feedcomp.*N***.in** float in
Reference value

feedcomp.*N***.enable** bit in
Turn compensation on or off

feedcomp.*N***.vel** float in
Current velocity

PARAMETERS

feedcomp.*N***.feed** float rw
Feed rate reference value

NOTES

Note that if enable is false, out = in

LICENSE

GPL

NAME

flipflop − D type flip-flop

SYNOPSIS

loadrt flipflop [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

flipflop.*N*

PINS

flipflop.*N***.data** bit in
: data input

flipflop.*N***.clk** bit in
: clock, rising edge writes data to out

flipflop.*N***.set** bit in
: when true, force out true

flipflop.*N***.reset** bit in
: when true, force out false; overrides set

flipflop.*N***.out** bit io
: output

LICENSE

GPL

NAME

freqgen − software step pulse generation

OBSOLETE - see **stepgen**'s 'ctrl_type=v' option.

SYNOPSIS

loadrt freqgen step_type=*type0*[,*type1*...]

DESCRIPTION

freqgen is used to control stepper motors. The maximum step rate depends on the CPU and other factors, and is usually in the range of 10KHz to 50KHz. If higher rates are needed, a hardware step generator is a better choice.

freqgen runs the motor at a commanded velocity, subject to acceleration and velocity limits. It does not directly control position.

freqgen can control a maximum of eight motors. The number of motors/channels actually loaded depends on the number of *type* values given. The value of each *type* determines the outputs for that channel. **freqgen** supports 15 possible step types.

By far the most common step type is '0', standard step and direction. Others include up/down, quadrature, and a wide variety of three, four, and five phase patterns that can be used to directly control some types of motor windings. (When used with appropriate buffers of course.)

Some of the stepping types are described below, but for more details (including timing diagrams) see the **stepgen** section of the HAL reference manual.

type 0: step/dir
: Two pins, one for step and one for direction. **make-pulses** must run at least twice for each step (once to set the step pin true, once to clear it). This limits the maximum step rate to half (or less) of the rate that can be reached by types 2-14. The parameters **steplen** and **stepspace** can further lower the maximum step rate. Parameters **dirsetup** and **dirhold** also apply to this step type.

type 1: up/down
: Two pins, one for 'step up' and one for 'step down'. Like type 0, **make-pulses** must run twice per step, which limits the maximum speed.

type 2: quadrature
: Two pins, phase-A and phase-B. For forward motion, A leads B. Can advance by one step every time **make-pulses** runs.

type 3: three phase, full step
: Three pins, phase-A, phase-B, and phase-C. Three steps per full cycle, then repeats. Only one phase is high at a time - for forward motion the pattern is A, then B, then C, then A again.

type 4: three phase, half step
: Three pins, phases A through C. Six steps per full cycle. First A is high alone, then A and B together, then B alone, then B and C together, etc.

types 5 through 8: four phase, full step
: Four pins, phases A through D. Four steps per full cycle. Types 5 and 6 are suitable for use with unipolar steppers, where power is applied to the center tap of each winding, and four open-collector transistors drive the ends. Types 7 and 8 are suitable for bipolar steppers, driven by two H-bridges.

types 9 and 10: four phase, half step
: Four pins, phases A through D. Eight steps per full cycle. Type 9 is suitable for unipolar drive, and type 10 for bipolar drive.

types 11 and 12: five phase, full step
: Five pins, phases A through E. Five steps per full cycle. See HAL reference manual for the patterns.

types 13 and 14: five phase, half step
: Five pins, phases A through E. Ten steps per full cycle. See HAL reference manual for the patterns.

FUNCTIONS

freqgen.make-pulses (no floating-point)
: Generates the step pulses, using information computed by **update-freq**. Must be called as frequently as possible, to maximize the attainable step rate and minimize jitter. Operates on all channels at once.

freqgen.capture-position (uses floating point)
: Captures position feedback value from the high speed code and makes it available on a pin for use elsewhere in the system. Operates on all channels at once.

freqgen.update-freq (uses floating point)
: Accepts a velocity command and converts it into a form usable by **make-pulses** for step generation. Operates on all channels at once.

PINS

freqgen.*N*.counts s32 out
: The current position, in counts, for channel *N*. Updated by **capture-position**.

freqgen.*N*.position-fb float out
: The current position, in length units (see parameter **position-scale**). Updated by **capture-position**.

freqgen.*N*.velocity float in (**freqgen** only)
: Commanded velocity, in length units per second (see parameter **velocity-scale**).

freqgen.*N*.step bit out (step type 0 only)
: Step pulse output.

freqgen.*N*.dir bit out (step type 0 only)
: Direction output: low for forward, high for reverse.

freqgen.*N*.up bit out (step type 1 only)
: Count up output, pulses for forward steps.

freqgen.*N*.down bit out (step type 1 only)
: Count down output, pulses for reverse steps.

freqgen.*N*.phase-A thru **phase-E** bit out (step types 2-14 only)
: Output bits. **phase-A** and **phase-B** are present for step types 2-14, **phase-C** for types 3-14, **phase-D** for types 5-14, and **phase-E** for types 11-14. Behavior depends on selected stepping type.

PARAMETERS

freqgen.*N*.frequency float ro
: The current step rate, in steps per second, for channel *N*.

freqgen.*N*.maxaccel float rw
: The acceleration/deceleration limit, in steps per second squared.

freqgen.*N*.maxfreq float rw (**freqgen** only)
: The maximum allowable velocity, in steps per second. If the requested maximum velocity cannot be reached with the current **make-pulses** thread period, it will be reset to the highest attainable value.

freqgen.*N*.position-scale float rw
: The scaling for position feedback, in steps per length unit.

freqgen.*N*.velocity-scale float rw
The scaling for the velocity command, in steps per length unit.

freqgen.*N*.rawcounts s32 ro
The position in counts, as updated by **make-pulses**. (Note: this is updated more frequently than the **counts** pin.)

freqgen.*N*.steplen u32 rw (step type 0 only)
The length of the step pulses, in **make-pulses** periods. Measured from rising edge to falling edge.

freqgen.*N*.stepspace u32 rw (step type 0 only)
The minimum space between step pulses, in **make-pulses** periods. Measured from falling edge to rising edge. The actual time depends on the step rate and can be much longer.

freqgen.*N*.dirsetup u32 rw (step type 0 only)
The minimum setup time from direction to step, in **make-pulses** periods. Measured from change of direction to rising edge of step.

freqgen.*N*.dirhold u32 rw (step type 0 only)
The minimum hold time of direction after step, in **make-pulses** periods. Measured from falling edge of step to change of direction.

BUGS

freqgen should have an **enable** pin.

freqgen's command pin should be called **velocity-cmd**, not **velocity**, for clarity and consistency with **stepgen**.

freqgen should use **maxvel**, not **maxfreq**. (In other words, the velocity limit should be scaled in length units per second, not steps per second. The scale parameter can be set to 1.0 if it is desired to work in steps instead of length units.)

freqgen's maxaccel parameter should be in length units per second squared, not steps per second squared, for consistency with **stepgen**.

freqgen should use **position-scale** for scaling both command and feedback, **velocity-scale** is redundant and should be eliminated.

Step type 1 (up/down) should respect the **steplen** and **stepspace** limits.

Timing parameters **steplen**, **stepspace**, **dirsetup**, and **dirhold** should be in nano-seconds, not **make-pulses** periods. That would allow the period to be changed without requiring the parameters to be recalculated.

All of these bugs have been fixed in **stepgen**. Only **stepgen** will continue to be maintained, since **freqgen** contains large amounts of code that duplicates code in **stepgen**. Since **stepgen** can provide the same functionality, there is no reason to maintain the duplicate code. **freqgen** may be eliminated at any time, and almost certainly **will** be eliminated for the version 2.2 release of EMC.

SEE ALSO

stepgen(9)

NAME

gantrykins − A kinematics module that maps one axis to multiple joints

SYNOPSIS

loadrt gantrykins coordinates=*axisletters*

Specifying gantry joint mapping via loadrt

The **coordinates=** parameter specifies the initial gantry joint mapping. Each axis letter is mapped to a joint, starting from 0. So **coordinates=XYYZ** maps the X axis to joint 0, the Y axis to joint 1 and 2, and the Z axis to joint 3. If not specified, the default mapping is **coordinates=XYZABC**. Coordinate letters may be specified in uppercase or lowercase.

A note about joints and axes

LinuxCNC makes a distinction between joints and axes: a joint is something controlled by a motor, and an axis is a coordinate you can move via G-code. You can also jog joints or jog axes.

A gantry has two joints controlling one axis, and this requires a bit of special care.

Homing always happens in joint mode (aka Free mode). The two joints of a gantry's axis must be homed together, so they must have the same [AXIS_n]HOME_SEQUENCE in the .ini file.

Jogging of a gantry must happen in world mode (aka Teleop mode). If you jog a gantry in joint mode (Free mode), you will move just one of the joints, and the gantry will rack. In contrast, if you jog a gantry in world mode (Teleop mode), it's the axis that jogs: linuxcnc will coordinate the motion of the two joints that make up the axis, both joints will move together, and the gantry will stay square.

The Axis GUI has provisions for jogging in joint mode (Free) and in world mode (Teleop). Use the "$" hotkey, or the View menu to switch between them.

Joint-mode (aka Free mode) supports continuous and incremental jogging. World-mode (aka Teleop mode) only supports continuous jogging.

KINEMATICS

In the inverse kinematics, each joint gets the value of its corresponding axis. In the forward kinematics, each axis gets the value of the highest numbered corresponding joint. For example, with **coordinates=XYYZ** the Y axis position comes from joint 2, not joint 1.

FUNCTIONS

None.

PINS

None.

PARAMETERS

gantrykins.joint-*N* (s32)

Specifies the axis mapped to joint *N*. The values 0 through 8 correspond to the axes XYZABCUVW. It is preferable to use the "coordinates=" parameter at loadrt-time rather than setting the joint-N parameters later, because the gantrykins module prints the joint-to-axis mapping at loadrt-time, and having that output correct is nice.

NOTES

gantrykins must be loaded before **motion**.

SEE ALSO

Kinematics section in the LinuxCNC documentation

LICENSE

GPL

NAME

gearchange − Select from one two speed ranges

SYNOPSIS

The output will be a value scaled for the selected gear, and clamped to the min/max values for that gear. The scale of gear 1 is assumed to be 1, so the output device scale should be chosen accordingly. The scale of gear 2 is relative to gear 1, so if gear 2 runs the spindle 2.5 times as fast as gear 1, scale2 should be set to 2.5.

FUNCTIONS

gearchange.*N* (requires a floating-point thread)

PINS

gearchange.*N***.sel** bit in
Gear selection input

gearchange.*N***.speed-in** float in
Speed command input

gearchange.*N***.speed-out** float out
Speed command to DAC/PWM

gearchange.*N***.dir-in** bit in
Direction command input

gearchange.*N***.dir-out** bit out
Direction output - possibly inverted for second gear

PARAMETERS

gearchange.*N***.min1** float rw (default: *0*)
Minimum allowed speed in gear range 1

gearchange.*N***.max1** float rw (default: *100000*)
Maximum allowed speed in gear range 1

gearchange.*N***.min2** float rw (default: *0*)
Minimum allowed speed in gear range 2

gearchange.*N***.max2** float rw (default: *100000*)
Maximum allowed speed in gear range 2

gearchange.*N***.scale2** float rw (default: *1.0*)
Relative scale of gear 2 vs. gear 1 Since it is assumed that gear 2 is "high gear", **scale2** must be greater than 1, and will be reset to 1 if set lower.

gearchange.*N***.reverse** bit rw (default: *0*)
Set to 1 to reverse the spindle in second gear

LICENSE

GPL

NAME

gladevcp − displays Virtual control Panels built with GTK / GLADE

SYNOPSIS

loadusr gladevcp [-c componentname0x*N*] [-g WxH+Xoffset+Yoffset0x*N*] [-H halcmdfile] [-x windowid] gladefile.glade

DESCRIPTION

gladevcp parses a glade file and displays the widgets in a window. Then calls gladevcp_makepins which again parses the gladefile looking for specific HAL widgets then makes HAL pins and sets up updating for them. The HAL component name defaults to the basename of the glade file. The -x option directs gladevcp to reparent itself under this X window id instead of creating its own toplevel window. The -H option passes an input file for halcmd to be run after the gladevcp component is initialized. This is used in Axis when running gladevcp under a tab with the EMBED_TAB_NAME/EMBED_TAB_COMMAND ini file feature.

gladevcp supports gtkbuilder or libglade files though some widgets are not fully supported in gtkbuilder yet.

ISSUES

For now system links need to be added in the glade library folders to point to our new widgets and catalog files. look in lib/python/gladevcp/READ_ME for details

NAME

gray2bin − convert a gray-code input to binary

SYNOPSIS

loadrt gray2bin [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

Converts a gray-coded number into the corresponding binary value

FUNCTIONS

gray2bin.*N*

PINS

gray2bin.*N***.in** u32 in
gray code in

gray2bin.*N***.out** u32 out
binary code out

AUTHOR

andy pugh

LICENSE

GPL

NAME

hm2_7i43 − LinuxCNC HAL driver for the Mesa Electronics 7i43 EPP Anything IO board with HostMot2 firmware.

SYNOPSIS

loadrt hm2_7i43 [ioaddr=*N*[,*N*...]] [ioaddr_hi=*N*[,*N*...]] [epp_wide=*N*[,*N*...]] [config=*"str[,str...]"*] [debug_epp=*N*[,*N*...]]

ioaddr [default: 0x378]
The base address of the parallel port.

ioaddr_hi [default: 0]
The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400.

epp_wide [default: 1]
Set to zero to disable the "wide EPP mode". "Wide" mode allows a 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this may not work on all EPP parallel ports.

config [default: ""]
HostMot2 config strings, described in the hostmot2(9) manpage.

debug_epp [default: 0]
Developer/debug use only! Enable debug logging of most EPP transfers.

DESCRIPTION

hm2_7i43 is a device driver that interfaces the Mesa 7i43 board with the HostMot2 firmware to the LinuxCNC HAL. Both the 200K and the 400K FPGAs are supported.

The driver talks with the 7i43 over the parallel port, not over USB. USB can be used to power the 7i43, but not to talk to it. USB communication with the 7i43 will not be supported any time soon, since USB has poor real-time qualities.

The driver programs the board's FPGA with firmware when it registers the board with the hostmot2 driver. The old bfload(1) firmware loading method is not used anymore. Instead the firmware to load is specified in the **config** modparam, as described in the hostmot2(9) manpage, in the *config modparam* section.

Some parallel ports require special initialization before they can be used. LinuxCNC provides a kernel driver that does this initialization called probe_parport. Load this driver before loading hm2_7i43, by putting "loadrt probe_parport" in your .hal file.

Jumper settings

To send the FPGA configuration from the PC, the board must be configured to get its firmware from the EPP port. To do this, jumpers W4 and W5 must both be down, ie toward the USB connector.

The board must be configured to power on whether or not the USB interface is active. This is done by setting jumper W7 up, ie away from the edge of the board.

Communicating with the board

The 7i43 communicates with the LinuxCNC computer over EPP, the Enhanced Parallel Port. This provides about 1 MBps of throughput, and the communication latency is very predictable and reasonably low.

The parallel port must support EPP 1.7 or EPP 1.9. EPP 1.9 is prefered, but EPP 1.7 will work too. The EPP mode of the parallel port is sometimes a setting in the BIOS.

Note that the popular "NetMOS" aka "MosChip 9805" PCI parport cards **do not work**. They do not meet the EPP spec, and cannot be reliably used with the 7i43. You have to find another card, sorry.

EPP is very reliable under normal circumstances, but bad cabling or excessively long cabling runs may

cause communication timeouts. The driver exports a parameter named hm2_7i43.<BoardNum>.io_error to inform HAL of this condition. When the driver detects an EPP timeout, it sets io_error to True and stops communicating with the 7i43 board. Setting io_error back to False makes the driver start trying to communicate with the 7i43 again.

Access to the EPP bus is not threadsafe: only one realtime thread may access the EPP bus.

SEE ALSO

hostmot2(9)

LICENSE

GPL

NAME

hm2_pci − LinuxCNC HAL driver for the Mesa Electronics PCI-based Anything IO boards, with HostMot2 firmware.

SYNOPSIS

loadrt hm2_pci [config=*"str[,str...]"***]**

config [default: ""]
HostMot2 config strings, described in the hostmot2(9) manpage.

DESCRIPTION

hm2_pci is a device driver that interfaces Mesa's PCI and PC-104/Plus based Anything I/O boards (with the HostMot2 firmware) to the LinuxCNC HAL.

The supported boards are: the 5i20, 5i22, and 5i23 (all on PCI); the 4i65 and 4i68 (on PC-104/Plus), and the 3x20 (on PCI Express, probably using a 6i68 or 7i68 carrier card).

The driver programs the board's FPGA with firmware when it registers the board with the hostmot2 driver. The firmware to load is specified in the **config** modparam, as described in the hostmot2(9) manpage, in the *config modparam* section.

SEE ALSO

hostmot2(9)

LICENSE

GPL

NAME

hostmot2 − LinuxCNC HAL driver for the Mesa Electronics HostMot2 firmware.

SYNOPSIS

See the config modparam section below for Mesa card configuration. Typically hostmot2 is loaded with no parameters unless debugging is required.

loadrt hostmot2 [debug_idrom=*N*] [debug_module_descriptors=*N*] [debug_pin_descriptors=*N*] [debug_modules=*N*]

debug_idrom [default: 0]
Developer/debug use only! Enable debug logging of the HostMot2 IDROM header.

debug_module_descriptors [default: 0]
Developer/debug use only! Enable debug logging of the HostMot2 Module Descriptors.

debug_pin_descriptors [default: 0]
Developer/debug use only! Enable debug logging of the HostMot2 Pin Descriptors.

debug_modules [default: 0]
Developer/debug use only! Enable debug logging of the HostMot2 Modules used.

DESCRIPTION

hostmot2 is a device driver that interfaces the Mesa HostMot2 firmware to the LinuxCNC HAL. This driver by itself does nothing, the boards that actually run the firmware require their own drivers before anything can happen. Currently drivers are available for the 5i20, 5i22, 5i23, 5i25, 3x20, 4i65, and 4i68 (all using the hm2_pci module) and the 7i43 (using the hm2_7i43 module).

The HostMot2 firmware provides modules such as encoders, PWM generators, step/dir generators, and general purpose I/O pins (GPIOs). These things are called "Modules". The firmware is configured, at firmware compile time, to provide zero or more instances of each of these Modules.

Board I/O Pins

The HostMot2 firmware runs on an FPGA board. The board interfaces with the computer via PCI, PC-104/Plus, or EPP, and interfaces with motion control hardware such as servos and stepper motors via I/O pins on the board.

Each I/O pin can be configured, at board-driver load time, to serve one of two purposes: either as a particular I/O pin of a particular Module instance (encoder, pwmgen, stepgen etc), or as a general purpose digital I/O pin. By default all Module instances are enabled, and all the board's pins are used by the Module instances.

The user can disable Module instances at board-driver load time, by specifying a hostmot2 config string modparam. Any pins which belong to Module instances that have been disabled automatically become GPIOs.

All IO pins have some HAL presence, whether they belong to an active module instance or are full GPIOs. GPIOs can be changed (at run-time) between inputs, normal outputs, and open drains, and have a flexible HAL interface. IO pins that belong to active Module instances are constrained by the requirements of the owning Module, and have a more limited interface in HAL. This is described in the General Purpose I/O section below.

config modparam

All the board-driver modules (hm2_pci and hm2_7i43) accept a load-time modparam of type string array, named "config". This array has one config string for each board the driver should use. Each board's config string is passed to and parsed by the hostmot2 driver when the board-driver registers the board.

The config string can contain spaces, so it is usually a good idea to wrap the whole thing in double-quotes (the " character).

The comma character (,) separates members of the config array from each other.

For example, if your control computer has one 5i20 and one 5i23 you might load the hm2_pci driver with a HAL command (in halcmd) something like this:

loadrt hm2_pci config="firmware=hm2/5i20/SVST8_4.BIT num_encoders=3 num_pwmgens=3 num_stepgens=3,firmw

Note: this assumes that the hm2_pci driver detects the 5i20 first and the 5i23 second. If the detection order does not match the order of the config strings, the hostmot2 driver will refuse to load the firmware and the board-driver (hm2_pci or hm2_7i43) will fail to load. To the best of my knowledge, there is no way to predict the order in which PCI boards will be detected by the driver, but the detection order will be consistent as long as PCI boards are not moved around. Best to try loading it and see what the detection order is.

The format of each board's config string is:

[firmware=*F*] [num_encoders=*N*] [num_resolvers=*N*] [num_pwmgens=*N*] [num_3pwmgens=*N*] [num_stepgens=*N*] [sserial_port_0=00000000] [num_leds=*N*] [enable_raw]

firmware [optional]
: Load the firmware specified by F into the FPGA on this board. If no "**firmware=***F*" string is specified, the FPGA will not be programmed, and had better have a valid configuration already.
The 5i25 comes pre-programmed with firmware and no "firmware=" string should be used. It can be loaded, if desired with no config string at all.

  The requested firmware F is fetched by udev. udev searches for the firmware in the system's firmware search path, usually /lib/firmware. F typically has the form "hm2/<BoardType>/file.bit"; a typical value for F might be "hm2/5i20/SVST8_4.BIT". The hostmot2 firmware files are supplied by the hostmot2-firmware packages, available from linuxcnc.org.

num_encoders [optional, default: -1]
: Only enable the first N encoders. If N is -1, all encoders are enabled. If N is 0, no encoders are enabled. If N is greater than the number of encoders available in the firmware, the board will fail to register.

num_resolvers [optional, default: -1]
: Only enable the first N resolvers. If N = -1 then all resolvers are enabled. This module does not work with generic resolvers (unlike the encoder module which works with any encoder). At the time of writing the Hostmot2 Resolver function only works with the Mesa 7i49 card.

num_pwmgens [optional, default: -1]
: Only enable the first N pwmgens. If N is -1, all pwmgens are enabled. If N is 0, no pwmgens are enabled. If N is greater than the number of pwmgens available in the firmware, the board will fail to register.

num_3pwmgens [optional, default: -1]
: Only enable the first N Three-phase pwmgens. If N is -1, all 3pwmgens are enabled. If N is 0, no pwmgens are enabled. If N is greater than the number of pwmgens available in the firmware, the board will fail to register.

num_stepgens [optional, default: -1]
: Only enable the first N stepgens. If N is -1, all stepgens are enabled. If N is 0, no stepgens are enabled. If N is greater than the number of stepgens available in the firmware, the board will fail to register.

sserial_port_N (N = 0 .. 3) [optional, default: 00000000 for all ports]

Up to 32 Smart Serial devices can be connected to a Mesa Anything IO board depending on the firmware used and the number of physical connections on the board. These are arranged in 1-4 ports of 1 to 8 channels.

Some Smart Serial (SSLBP) cards offer more than one load-time configuration, for example all inputs, or all outputs, or offering additional analogue input on some digital pins.

To set the modes for port 0 use, for example **sserial_port_0=0120xxxx**

A ’0’in the string sets the corresponding port to mode 0, 1 to mode 1, and so on up to mode 9. An "x" in any position disables that channel and makes the corresponding FPGA pins available as GPIO.

The string can be up to 8 characters long, and if it defines more modes than there are channels on the port then the extras are ignored. Channel numbering is left to right so the example above would set sserial device 0.0 to mode 0, 0.2 to mode2 and disable channels 0.4 onwards.

The sserial driver will auto-detect connected devices, no further configuration should be needed. Unconnected channels will default to GPIO, but the pin values will vary semi-randomly during boot when card-detection runs, to it is best to actively disable any channel that is to be used for GPIO.

num_leds [optional, default: -1]

Only enable the first N of the LEDs on the FPGA board. If N is -1, then HAL pins for all the LEDs will be created. If N=0 then no pins will be added.

enable_raw [optional]

If specified, this turns on a raw access mode, whereby a user can peek and poke the firmware from HAL. See Raw Mode below.

encoder

Encoders have names like "hm2_<*BoardType*>.<*BoardNum*>.encoder.<*Instance*>". "Instance" is a two-digit number that corresponds to the HostMot2 encoder instance number. There are ’num_encoders’ instances, starting with 00.

So, for example, the HAL pin that has the current position of the second encoder of the first 5i20 board is: hm2_5i20.0.encoder.01.position (this assumes that the firmware in that board is configured so that this HAL object is available)

Each encoder uses three or four input IO pins, depending on how the firmware was compiled. Three-pin encoders use A, B, and Index (sometimes also known as Z). Four-pin encoders use A, B, Index, and Index-mask.

The hm2 encoder representation is similar to the one described by the Canonical Device Interface (in the HAL General Reference document), and to the software encoder component. Each encoder instance has the following pins and parameters:

Pins:

(s32 out) count

Number of encoder counts since the previous reset.

(float out) position

Encoder position in position units (count / scale).

(float out) velocity
Estimated encoder velocity in position units per second.

(bit in) reset
When this pin is TRUE, the count and position pins are set to 0. (The value of the velocity pin is not affected by this.) The driver does not reset this pin to FALSE after resetting the count to 0, that is the user's job.

(bit in/out) index-enable
When this pin is set to True, the count (and therefore also position) are reset to zero on the next Index (Phase-Z) pulse. At the same time, index-enable is reset to zero to indicate that the pulse has occurred.

(s32 out) rawcounts
Total number of encoder counts since the start, not adjusted for index or reset.

Parameters:

(float r/w) scale
Converts from 'count' units to 'position' units.

(bit r/w) index-invert
If set to True, the rising edge of the Index input pin triggers the Index event (if index-enable is True). If set to False, the falling edge triggers.

(bit r/w) index-mask
If set to True, the Index input pin only has an effect if the Index-Mask input pin is True (or False, depending on the index-mask-invert pin below).

(bit r/w) index-mask-invert
If set to True, Index-Mask must be False for Index to have an effect. If set to False, the Index-Mask pin must be True.

(bit r/w) counter-mode
Set to False (the default) for Quadrature. Set to True for Step/Dir (in which case Step is on the A pin and Dir is on the B pin).

(bit r/w) filter
If set to True (the default), the quadrature counter needs 15 clocks to register a change on any of the three input lines (any pulse shorter than this is rejected as noise). If set to False, the quadrature counter needs only 3 clocks to register a change. The encoder sample clock runs at 33 MHz on the PCI AnyIO cards and 50 MHz on the 7i43.

(float r/w) vel-timeout
When the encoder is moving slower than one pulse for each time that the driver reads the count from the FPGA (in the hm2_read() function), the velocity is harder to estimate. The driver can wait several iterations for the next pulse to arrive, all the while reporting the upper bound of the encoder velocity, which can be accurately guessed. This parameter specifies how long to wait for the next pulse, before reporting the encoder stopped. This parameter is in seconds.

resolver

Resolvers have names like hm2_<*BoardType*>.<*BoardNum*>.resolver.<*Instance*>. <Instance is a 2-digit number, which for the 7i49 board will be between 00 and 05. This function only works with the Mesa Resolver interface boards (of which the 7i49 is the only example at the time of writing). This board uses an SPI interface to the FPGA card, and will only work with the correct firmware. The pins allocated will be listed in the dmesg output, but are unlikely to be usefully probed with HAL tools.

Pins:

(float, out) angle
: This pin indicates the angular position of the resolver. It is a number between 0 and 1 for each electrical rotation.

(float, out) position
: Calculated from the number of complete and partial revolutions since startup, reset, or index-reset multiplied by the scale parameter.

(float, out) velocity
: Calculated from the rotational velocity and the velocity-scale parameter. The default scale is electrical rotations per second.

(s32, out) count
: This pins outputs a simulated encoder count at 2^24 counts per rev (16777216 counts).

(s32, out) rawcounts
: This is identical to the counts pin, except it is not reset by the 'index' or 'reset' pins. This is the pin which would be linked to the bldc HAL component if the resolver was being used to commutate a motor.

(bit, in) reset
: Resets the position and counts pins to zero immediately.

(bit, in/out) index-enable
: When this pin is set high the position and counts pins will be reset the next time the resolver passes through the zero position. At the same time the pin is driven low to indicate to connected modules that the index has been seen, and that the counters have been reset.

(bit, out) error
: Indicates an error in the particular channel. If this value is "true" then the reported position and velocity are invalid.

Parameters:

(float, read/write) scale
: The position scale, in machine units per resolver electrical revolution.

(float, read/write) velocity-scale
: The conversion factor between resolver rotation speed and machine velocity. A value of 1 will typically give motor speed in rps, a value of 0.01666667 will give (approximate) RPM.

(float, read/write) excitation-khz

This pin sets the excitation frequency for the resolver. This pin is module-level rather than instance-level as all resolvers share the same excitation frequency.

Valid values are 10 (˜10kHz), 5 (˜5kHz) and 2.5 (˜2.5kHz). The actual frequency depends on the FPGA frequency, and they correspond to CLOCK_LOW/5000, CLOCK_LOW/10000 and CLOCK_LOW/20000 respectively. The parameter will be set to the closest available of the three frequencies.

A value of -1 (the default) indicates that the current setting should be retained.

pwmgen

pwmgens have names like "hm2_<*BoardType*>.<*BoardNum*>.pwmgen.<*Instance*>". "Instance" is a two-digit number that corresponds to the HostMot2 pwmgen instance number. There are 'num_pwmgens' instances, starting with 00.

So, for example, the HAL pin that enables output from the fourth pwmgen of the first 7i43 board is: hm2_7i43.0.pwmgen.03.enable (this assumes that the firmware in that board is configured so that this HAL object is available)

In HM2, each pwmgen uses three output IO pins: Not-Enable, Out0, and Out1.

The function of the Out0 and Out1 IO pins varies with output-type parameter (see below).

The hm2 pwmgen representation is similar to the software pwmgen component. Each pwmgen instance has the following pins and parameters:

Pins:

(bit input) enable

If true, the pwmgen will set its Not-Enable pin false and output its pulses. If 'enable' is false, pwmgen will set its Not-Enable pin true and not output any signals.

(float input) value

The current pwmgen command value, in arbitrary units.

Parameters:

(float rw) scale

Scaling factor to convert 'value' from arbitrary units to duty cycle: dc = value / scale. Duty cycle has an effective range of -1.0 to +1.0 inclusive, anything outside that range gets clipped. The default scale is 1.0.

(s32 rw) output-type

This emulates the output_type load-time argument to the software pwmgen component. This parameter may be changed at runtime, but most of the time you probably want to set it at startup and then leave it alone. Accepted values are 1 (PWM on Out0 and Direction on Out1), 2 (Up on Out0 and Down on Out1), 3 (PDM mode, PDM on Out0 and Dir on Out1), and 4 (Direction on Out0 and PWM on Out1, "for locked antiphase").

In addition to the per-instance HAL Parameters listed above, there are a couple of HAL Parameters that affect all the pwmgen instances:

(u32 rw) pwm_frequency

This specifies the PWM frequency, in Hz, of all the pwmgen instances running in the PWM modes (modes 1 and 2). This is the frequency of the variable-duty-cycle wave. Its effective range is from 1 Hz up to 193 kHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything IO board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 193 kHz max PWM frequency. Other boards may have different clocks, resulting in different max PWM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. Frequencies below about 5 Hz are not terribly accurate, but above 5 Hz they're pretty close. The default pwm_frequency is 20,000 Hz (20 kHz).

(u32 rw) pdm_frequency

This specifies the PDM frequency, in Hz, of all the pwmgen instances running in PDM mode (mode 3). This is the "pulse slot frequency"; the frequency at which the pdm generator in the AnyIO board chooses whether to emit a pulse or a space. Each pulse (and space) in the PDM pulse train has a duration of 1/pdm_frequency seconds. For example, setting the pdm_frequency to 2e6 (2 MHz) and the duty cycle to 50% results in a 1 MHz square wave, identical to a 1 MHz PWM signal with 50% duty cycle. The effective range of this parameter is from about 1525 Hz up to just under 100 MHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything IO board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 100 Mhz max PDM frequency. Other boards may have different clocks, resulting in different max PDM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. The default pdm_frequency is 20,000 Hz (20 kHz).

3ppwmgen

Three-Phase PWM generators (3pwmgens) are intended for controlling the high-side and low-side gates in a 3-phase motor driver. The function is included to support the Mesa motor controller daughter-cards but can be used to control an IGBT or similar driver directly. 3pwmgens have names like "hm2_<*BoardType*>.<*BoardNum*>.3pwmgen.<*Instance*>" where <Instance> is a 2-digit number. There will be num_3pwmgens instances, starting at 00. Each instance allocates 7 output and one input pins on the Mesa card connectors. Outputs are: PWM A, PWM B, PWM C, /PWM A, /PWM B, /PWM C, Enable. The first three pins are the high side drivers, the second three are their complementary low-side drivers. The enable bit is intended to control the servo amplifier. The input bit is a fault bit, typically wired to over-current detection. When set the PWM generator is disabled. The three phase duty-cycles are individually controllable from -Scale to +Scale. Note that 0 corresponds to a 50% duty cycle and this is the inialization value.

Pins:

(float input) A-value, B-value, C-value: The PWM command value for each phase, limited to +/- "scale". Defaults to zero which is 50% duty cycle on high-side and low-sidepins (but see the "deadtime" parameter)

(bit input) enable

When high the PWM is enabled as long as the fault bit is not set by the external fault input pin. When low the PWM is disabled, with both high- side and low-side drivers low. This is not the same as 0 output (50% duty cycle on both sets of pins) or negative full scale (where the low side drivers are "on" 100% of the time)

(bit output) fault

Indicates the status of the fault bit. This output latches high once set by the physical fault pin until the "enable" pin is set to high.

Parameters:

(u32 rw) deadtime
: Sets the dead-time between the high-side driver turning off and the low-side driver turning on and vice-versa. Deadtime is subtracted from on time and added to off time symmetrically. For example with 20 kHz PWM (50 uSec period), 50% duty cycle and zero dead time, the PWM and NPWM outputs would be square waves (NPWM being inverted from PWM) with high times of 25 uS. With the same settings but 1 uS of deadtime, the PWM and NPWM outputs would both have high times of 23 uS (25 - (2X 1 uS), 1 uS per edge). The value is specified in nS and defaults to a rather conservative 5000nS. Setting this parameter to too low a value could be both expensive and dangerous as if both gates are open at the same time there is effectively a short circuit accross the supply.

(float rw) scale
: Sets the half-scale of the specified 3-phase PWM generator. PWM values from -scale to +scale are valid. Default is +/- 1.0

(bit rw) fault-invert
: Sets the polarity of the fault input pin. A value of 1 means that a fault is triggered with the pin high, and 0 means that a fault it triggered when the pin is pulled low. Default 0, fault = low so that the PWM works with the fault pin unconnected.

(u32 rw) sample-time
: Sets the time during the cycle when an ADC pulse is generated. 0 = start of PWM cycle and 1 = end. Not currently useful to LinuxCNC. Default 0.5.

  In addition the per-instance parameters above there is the following parameter that affects all instances

(u32 rw) frequency
: Sets the master PWM frequency. Maximum is approx 48kHz, minimum is 1kHz. Defaults to 20kHz.

stepgen

stepgens have names like "hm2_<*BoardType*>.<*BoardNum*>.stepgen.<*Instance*>". "Instance" is a two-digit number that corresponds to the HostMot2 stepgen instance number. There are 'num_stepgens' instances, starting with 00.

So, for example, the HAL pin that has the current position feedback from the first stepgen of the second 5i22 board is: hm2_5i22.1.stepgen.00.position-fb (this assumes that the firmware in that board is configured so that this HAL object is available)

Each stepgen uses 2 IO pins. The signals on these pins depends on the step_type parameter (described below).

The stepgen representation is modeled on the stepgen software component. Each stepgen instance has the following pins and parameters:

Pins:

(float input) position-cmd
: Target position of stepper motion, in arbitrary position units. This pin is only used when the stepgen is in position control mode (control-type=0).

(float input) velocity-cmd
: Target velocity of stepper motion, in arbitrary position units per second. This pin is only used when the stepgen is in velocity control mode (control-type=1).

(s32 output) counts
: Feedback position in counts (number of steps).

(float output) position-fb
: Feedback position in arbitrary position units. This is similar to "counts/position_scale", but has finer than step resolution.

(float output) velocity-fb
: Feedback velocity in arbitrary position units per second.

(bit input) enable
: This pin enables the step generator instance. When True, the stepgen instance works as expected. When False, no steps are generated and velocity-fb goes immediately to 0. If the stepgen is moving when enable goes false it stops immediately, without obeying the maxaccel limit.

(bit input) control-type
: Switches between position control mode (0) and velocity control mode (1). Defaults to position control (0).

  Parameters:

(float r/w) position-scale
: Converts from counts to position units. position = counts / position_scale

(float r/w) maxvel
: Maximum speed, in position units per second. If set to 0, the driver will always use the maximum possible velocity based on the current step timings and position-scale. The max velocity will change if the step timings or position-scale changes. Defaults to 0.

(float r/w) maxaccel
: Maximum acceleration, in position units per second per second. Defaults to 1.0. If set to 0, the driver will not limit its acceleration at all - this requires that the position-cmd or velocity-cmd pin is driven in a way that does not exceed the machine's capabilities. This is probably what you want if you're going to be using the LinuxCNC trajectory planner to jog or run G-code.

(u32 r/w) steplen
: Duration of the step signal, in nanoseconds.

(u32 r/w) stepspace
: Minimum interval between step signals, in nanoseconds.

(u32 r/w) dirsetup
: Minimum duration of stable Direction signal before a step begins, in nanoseconds.

(u32 r/w) dirhold
Minimum duration of stable Direction signal after a step ends, in nanoseconds.

(u32 r/w) step_type
Output format, like the step_type modparam to the software stegen(9) component. 0 = Step/Dir, 1 = Up/Down, 2 = Quadrature. In Quadrature mode (step_type=2), the stepgen outputs one complete Gray cycle (00 â 01 â 11 â 10 â 00) for each "step" it takes.

Smart Serial Interface

The Smart Serial Interface allows up to 32 different devices such as the Mesa 8i20 2.2kW 3-phase drive or 7i64 48-way IO cards to be connected to a single FPGA card. The driver auto-detects the connected hardware port, channel and device type. Devices can be connected in any order to any active channel of an active port. (see the config modparam definition above).

In addition to the per-channel/device pins detailed below there are three per-port pins, three parameters and a further 7 pins shared between all ports used for changing on-device settings.

Pins:

(bit, in) .sserial.port-N.run: Enables the specific Smart Serial module. Setting this pin low will disable all boards on the port and puts the port in a pass-through mode where device parameter setting is possible. This pin defaults to TRUE and can be left unconnected. However, toggling the pin low-to-high will re-enable a faulted drive so the pin could usefully be connected to the iocontrol.0.user-enable-out pin.

(u32, ro) .run_state: Shows the state of the sserial communications state-machine. This pin will generally show a value of 0x03 in normal operation, 0x07 in setup mode and 0x00 when the "run" pin is false.

(u32, ro) .error-count: Indicates the state of the Smart Serial error handler, see the parameters sections for more details.

(u32, in) .sserial.port: When programming device parameters, this pin sets the active port.

(u32, in) .sserial.channel: This sets the active channel on the port defined above.

(u32, in) .sserial.parameter: The two LSBs of this pin determine the parameter address on the device connected to the channel and port defined above which will be read from or written to. Byte 2 is used to pass the data type for non-volatile writes, and byte 3 passes the access type flags. For example 0xEC010002 is the code to use to set 8i20 NVMAXCURRENT.

(u32, in) .sserial.value: This pin is used to set the value to be written to a device parameter. It can also be used to read-back a value from the device, but as it is an input pin the read-back value is only useful to humans.

(bit, in) .sserial.read: Starts the parameter read cycle. On completion the pin will return to zero (if not connected in HAL)

(bit, in) .sserial.write: Starts the parameter write cycle. On completion the pin will return to zero (if not connected in HAL)

(u32, out) .sserial.state: Shows the state of the read or write process (which typically takes many servo cycles). See the source code if the precise meanings are important to your application.

Parameters:

(u32 r/w) .fault-inc: Any over-run or handshaking error in the SmartSerial communications will increment the .fault-count pin by the amount specified by this parameter. Default = 10.

(u32 r/w) .fault-dec: Every successful read/write cycle decrements the fault counter by this amount. Default = 1.

(u32 r/w) .fault-lim: When the fault counter reaches this threshold the Smart Serial interface on the corresponding port will be stopped and an error printed in dmesg. Together these three pins allow for control over the degree of fault- tolerance allowed in the interface. The default values mean that if more than one transaction in ten fails, more than 20 times, then a hard error will be raised. If the increment were to be set to zero then no error would ever be raised, and the system would carry on regardless. Conversely setting decrement to zero, threshold to 1 and limit to 1 means that absolutely no errors will be tolerated. (This structure is copied directly from vehicle ECU practice)

The other pins and parameters created in HAL depend on the devices detected.

8i20 The 8i20 is a 2.2kW three-phase drive for brushless DC motors and AC servo motors. 8i20 pins and parameters have names like "hm2_<*BoardType*>.<*BoardNum*>.8i20.<*PortNum*>.<*ChanNum*>.<*Pin*>", for example "hm2_5i23.0.8i20.1.3.current" would set the phase current for the drive connected to the fourth channel of the second sserial port of the first 5i23 board. Note that the sserial ports do not necessarily correlate in layout or number to the physical ports on the card.

Pins:

(bit, in) amp-enable
: Set this pin high to enable the drive. With the pin low communications are active but the phase angle and current are set to zero.

(float in) angle
: The rotor angle of the motor in fractions of a full **phase** revolution. An angle of 0.5 indicates that the motor is half a turn / 180 degrees / π radians from the zero position. The zero position is taken to be the position that the motor adopts under no load with a poitive voltage applied to the A (or U) phase and both B and C (or V and W) connected to -V or 0V. A 6 pole motor will have 3 zero positions per physical rotation. Note that the 8i20 drive automatically adds the phase lead/lag angle, and that this pin should see the raw rotor angle. There is a HAL module (bldc) which handles the complexity of differing motor and drive types.

(float, in) current
: The phase current command to the drive. This is scaled from -1 to +1 for forwards and reverse maximum currents. The absolute value of the current is set by the max_current parameter.

(float, ro) voltage
: The drive bus voltage in V. This will tend to show 25.6V when the drive is unpowered and the drive will not operate below about 50V.

(float, ro) temp
: The temperature of the driver in degrees C.

(u32, ro) fault
: The fault status of the drive. See the 8i20 manual for a detailed description.

(u32, ro) status
The operating status of the drive. See the 8i20 manual for more details.

(u32, ro) comms
The communication status of the drive. See the manual for more details.

Parameters:

(float, rw) max_current
Sets the maximum drive current in Amps. The default value is the maximum current programmed into the drive EEPROM. The value must be positive, and an error will be raised if a current in excess of the drive maximum is requested.

(u32, ro) serial_number
The serial number of the connected drive. This is also shown on the label on the drive.

7i64 The 7i64 is a 24-input 24-output IO card. 7i64 pins and parameters have names like "hm2_<*BoardType*>.<*BoardNum*>.7i64. <PortNum>.<ChanNum>.<Pin>", for example hm2_5i23.0.7i63.1.3.digout.01.out

Pins: (bit, in) .digout.NN.out: Writing a 1 or TRUE to this pin will enable output driver NN. Note that the outputs are drivers (switches) rather than voltage outputs. The LED adjacent to the connector on the board shows the status.

(bit, out) .digin.NN.in: The value of input NN. Note that the inputs are isolated and both pins of each input must be connected (typically to signal and the ground of the signal. This need not be the ground of the board.)

(bit, out) .digin.NN.in-not: An inverted copy of the corresponding input.

(float, out) .adcin.00.in & .adcin.01.in: The two analogue inputs (0 to 3.3V) on the board.

Parameters: (bit, rw) .digout.NN.invert: Setting this parameter to 1 / TRUE will invert the output value, such that writing 0 to .gpio.NN.out will enable the output and vice-versa.

General Purpose I/O

I/O pins on the board which are not used by a module instance are exported to HAL as "full" GPIO pins. Full GPIO pins can be configured at run-time to be inputs, outputs, or open drains, and have a HAL interface that exposes this flexibility. IO pins that are owned by an active module instance are constrained by the requirements of the owning module, and have a restricted HAL interface.

GPIOs have names like "hm2_<*BoardType*>.<*BoardNum*>.gpio.<*IONum*>". IONum is a three-digit number. The mapping from IONum to connector and pin-on-that-connector is written to the syslog when the driver loads, and it's documented in Mesa's manual for the Anything I/O boards.

So, for example, the HAL pin that has the current inverted input value read from GPIO 012 of the second 7i43 board is: hm2_7i43.1.gpio.012.in-not (this assumes that the firmware in that board is configured so that this HAL object is available)

The HAL parameter that controls whether the last GPIO of the first 5i22 is an input or an output is: hm2_5i22.0.gpio.095.is_output (this assumes that the firmware in that board is configured so that this HAL object is available)

The hm2 GPIO representation is modeled after the Digital Inputs and Digital Outputs described in the Canonical Device Interface (part of the HAL General Reference document). Each GPIO can have the following HAL Pins:

(bit out) in & in_not: State (normal and inverted) of the hardware input pin. Both full GPIO pins and IO pins used as inputs by active module instances have these pins.

(bit in) out
: Value to be written (possibly inverted) to the hardware output pin. Only full GPIO pins have this pin.

  Each GPIO can have the following Parameters:

(bit r/w) is_output
: If set to 0, the GPIO is an input. The IO pin is put in a high-impedance state (weakly pulled high), to be driven by other devices. The logic value on the IO pin is available in the "in" and "in_not" HAL pins. Writes to the "out" HAL pin have no effect. If this parameter is set to 1, the GPIO is an output; its behavior then depends on the "is_opendrain" parameter. Only full GPIO pins have this parameter.

(bit r/w) is_opendrain
: This parameter only has an effect if the "is_output" parameter is true. If this parameter is false, the GPIO behaves as a normal output pin: the IO pin on the connector is driven to the value specified by the "out" HAL pin (possibly inverted), and the value of the "in" and "in_not" HAL pins is undefined. If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the "out" HAL pin drives the IO pin low, writing 1 to the "out" HAL pin puts the IO pin in a high-impedance state. In this high-impedance state the IO pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the IO pin is available on the "in" and "in_not" pins. Only full GPIO pins and IO pins used as outputs by active module instances have this parameter.

(bit r/w) invert_output
: This parameter only has an effect if the "is_output" parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the "out" HAL pin. Only full GPIO pins and IO pins used as outputs by active module instances have this parameter.

led

Creates HAL pins for the LEDs on the FPGA board.

Pins:

(bit in) CR<NN>
: The pins are numbered from CR01 upwards with the name corresponding to the PCB silkscreen. Setting the bit to "true" or 1 lights the led.

Watchdog

The HostMot2 firmware may include a watchdog Module; if it does, the hostmot2 driver will use it. The HAL representation of the watchdog is named "hm2_<**BoardType**>.<**BoardNum**>.watchdog".

The watchdog starts out asleep and inactive. Once you access the board the first time by running any the hm2 HAL functions read(), write(), or pet_watchdog() (see below), the watchdog wakes up. From them on it must be petted periodically or it will bite. Pet the watchdog by running the pet_watchdog() HAL function.

When the watchdog bites, all the board's I/O pins are disconnected from their Module instances and become high-impedance inputs (pulled high), and all communication with the board stops. The state of the HostMot2 firwmare modules is not disturbed (except for the configuration of the IO Pins). Encoder instances keep counting quadrature pulses, and pwm- and step-generators keep generating signals (which are *not* relayed to the motors, because the IO Pins have become inputs).

Resetting the watchdog (by clearing the has_bit pin, see below) resumes communication and resets the I/O pins to the configuration chosen at load-time.

If the firmware includes a watchdog, the following HAL objects will be exported:

Pins:

(bit in/out) has_bit
: True if the watchdog has bit, False if the watchdog has not bit. If the watchdog has bit and the has_bit bit is True, the user can reset it to False to resume operation.

Parameters:

(u32 read/write) timeout_ns
: Watchdog timeout, in nanoseconds. This is initialized to 5,000,000 (5 milliseconds) at module load time. If more than this amount of time passes between calls to the pet_watchdog() function, the watchdog will bite.

Functions:

pet_watchdog(): Calling this function resets the watchdog timer (postponing the watchdog biting until timeout_ns nanoseconds later).

Raw Mode

If the "enable_raw" config keyword is specified, some extra debugging pins are made available in HAL. The raw mode HAL pin names begin with "hm2_<*BoardType*>.<*BoardNum*>.raw".

With Raw mode enabled, a user may peek and poke the firmware from HAL, and may dump the internal state of the hostmot2 driver to the syslog.

Pins:

(u32 in) read_address
: The bottom 16 bits of this is used as the address to read from.

(u32 out) read_data
: Each time the hm2_read() function is called, this pin is updated with the value at .read_address.

(u32 in) write_address
: The bottom 16 bits of this is used as the address to write to.

(u32 in) write_data
: This is the value to write to .write_address.

(bit in) write_strobe
: Each time the hm2_write() function is called, this pin is examined. If it is True, then value in .write_data is written to the address in .write_address, and .write_strobe is set back to False.

(bit in/out) dump_state
: This pin is normally False. If it gets set to True the hostmot2 driver will write its representation of the board's internal state to the syslog, and set the pin back to False.

Setting up Smart Serial devices

If a Smart Serial port is stopped, it is then possible to read and set the onboard firmware settings using "setp" commands from within halrun or with halcmd. The hostmot read function needs to be attached to a realtime thread to allow parameters to be read and set, but the write thread is not needed. A typical command-line sequence to read the max current setting of an 8i20 on channel 1 of port 0 would be:

```
$>halrun

halcmd: loadrt hostmot2
halcmd: loadrt hm2_pci config="firmware=hm2/5i23/svss8_44 sserial_port_0=x0xxxxxxx"
halcmd: loadrt threads
halcmd: addf hm2_5i23.0.read thread1
halcmd: addf hm2_5i23.0.write thread1
halcmd: start
halcmd: setp hm2_5i23.0.sserial.0.port-0.run 0
halcmd: setp hm2_5i23.0.sserial.0.port 0
halcmd: setp hm2_5i23.0.sserial.0.channel 1
halcmd: setp hm2_5i23.0.sserial.0.0.parameter 0x8e8
halcmd: setp hm2_5i23.0.sserial.0.0.read 1
halcmd: show pin hm2_5i23.0.sserial.0.0.value
```

It is not expected that this would be a frequent task. The pin descriptions appear earlier in this document.

FUNCTIONS

hm2_<*BoardType*>**.**<*BoardNum*>**.read**
: This reads the encoder counters, stepgen feedbacks, and GPIO input pins from the FPGA.

hm2_<*BoardType*>**.**<*BoardNum*>**.write**
: This updates the PWM duty cycles, stepgen rates, and GPIO outputs on the FPGA. Any changes to configuration pins such as stepgen timing, GPIO inversions, etc, are also effected by this function.

hm2_<*BoardType*>**.**<*BoardNum*>**.pet-watchdog**
: Pet the watchdog to keep it from biting us for a while.

hm2_<*BoardType*>**.**<*BoardNum*>**.read_gpio**
: Read the GPIO input pins. Note that the effect of this function is a subset of the effect of the .read() function described above. Normally only .read() is used. The only reason to call this function is if you want to do GPIO things in a faster-than-servo thread. (This function is not available on the 7i43 due to limitations of the EPP bus.)

hm2_<*BoardType*>**.**<*BoardNum*>**.write_gpio**
: Write the GPIO control registers and output pins. Note that the effect of this function is a subset of the effect of the .write() function described above. Normally only .write() is used. The only reason to call this function is if you want to do GPIO things in a faster-than-servo thread. (This function is not available on the 7i43 due to limitations of the EPP bus.)

SEE ALSO

hm2_7i43(9)
hm2_pci(9)
Mesa's documentation for the Anything I/O boards, at <http://www.mesanet.com>

LICENSE

GPL

NAME

hypot − Three-input hypotenuse (Euclidean distance) calculator

SYNOPSIS

loadrt hypot [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

hypot.*N* (requires a floating-point thread)

PINS

hypot.*N***.in0** float in
hypot.*N***.in1** float in
hypot.*N***.in2** float in
hypot.*N***.out** float out
out = sqrt(in0ˆ2 + in1ˆ2 + in2ˆ2)

LICENSE

GPL

NAME

ilowpass − Low-pass filter with integer inputs and outputs

SYNOPSIS

loadrt ilowpass [count=*N*|**names=***name1*[**,***name2...*]**]**

DESCRIPTION

While it may find other applications, this component was written to create smoother motion while jogging with an MPG.

In a machine with high acceleration, a short jog can behave almost like a step function. By putting the **ilowpass** component between the MPG encoder **counts** output and the axis jog-counts input, this can be smoothed.

Choose **scale** conservatively so that during a single session there will never be more than about 2e9/**scale** pulses seen on the MPG. Choose **gain** according to the smoothing level desired. Divide the axis.*N*.jog-scale values by **scale**.

FUNCTIONS

ilowpass.*N* (requires a floating-point thread)

Update the output based on the input and parameters

PINS

ilowpass.*N***.in** s32 in

ilowpass.*N***.out** s32 out

out tracks **in***__*__**scale** through a low-pass filter of **gain** per period.

PARAMETERS

ilowpass.*N***.scale** float rw (default: *1024*)

A scale factor applied to the output value of the low-pass filter.

ilowpass.*N***.gain** float rw (default: *.5*)

Together with the period, sets the rate at which the output changes. Useful range is between 0 and 1, with higher values causing the input value to be tracked more quickly. For instance, a setting of 0.9 causes the output value to go 90% of the way towards the input value in each period

AUTHOR

Jeff Epler <jepler@unpythonic.net>

LICENSE

GPL

NAME

integ − Integrator with gain pin and windup limits

SYNOPSIS

loadrt integ [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

integ.*N* (requires a floating-point thread)

PINS

integ.*N***.in** float in
integ.*N***.gain** float in (default: *1.0*)
integ.*N***.out** float out
The discrete integral of ’gain * in’ since ’reset’ was deasserted

integ.*N***.reset** bit in
When asserted, set out to 0

integ.*N***.max** float in (default: *1e20*)
integ.*N***.min** float in (default: *-1e20*)

LICENSE

GPL

NAME

invert − Compute the inverse of the input signal

SYNOPSIS

The output will be the mathematical inverse of the input, ie **out** = 1/**in**. The parameter **deadband** can be used to control how close to 0 the denominator can be before the output is clamped to 0. **deadband** must be at least 1e-8, and must be positive.

FUNCTIONS

invert.*N* (requires a floating-point thread)

PINS

invert.*N***.in** float in
Analog input value

invert.*N***.out** float out
Analog output value

PARAMETERS

invert.*N***.deadband** float rw
The **out** will be zero if **in** is between **-deadband** and +**deadband**

LICENSE

GPL

NAME

joyhandle − sets nonlinear joypad movements, deadbands and scales

SYNOPSIS

loadrt joyhandle [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

The component **joyhandle** uses the following formula for a non linear joypad movements:

y = (scale * (a*xˆpower + b*x)) + offset

The parameters a and b are adjusted in such a way, that the function starts at (deadband,offset) and ends at (1,scale+offset).

Negative values will be treated point symetrically to origin. Values -deadband < x < +deadband will be set to zero.

Values x > 1 and x < -1 will be skipped to ±(scale+offset). Invert transforms the function to a progressive movement.

With power one can adjust the nonlinearity (default = 2). Default for deadband is 0.

Valid values are: power >= 1.0 (reasonable values are 1.x .. 4-5, take higher power-values for higher deadbands (>0.5), if you want to start with a nearly horizontal slope), 0 <= deadband < 0.99 (reasonable 0.1).

An additional offset component can be set in special cases (default = 0).

All values can be adjusted for each instance separately.

FUNCTIONS

joyhandle.*N* (requires a floating-point thread)

PINS

joyhandle.*N***.in** float in
joyhandle.*N***.out** float out

PARAMETERS

joyhandle.*N***.power** float rw (default: *2.0*)
joyhandle.*N***.deadband** float rw (default: *0.*)
joyhandle.*N***.scale** float rw (default: *1.*)
joyhandle.*N***.offset** float rw (default: *0.*)
joyhandle.*N***.inverse** bit rw (default: *0*)

LICENSE

GPL

NAME

kins − kinematics definitions for LinuxCNC

SYNOPSIS

loadrt trivkins

loadrt rotatekins

loadrt tripodkins

loadrt genhexkins

loadrt maxkins

loadrt genserkins

loadrt pumakins

loadrt scarakins

DESCRIPTION

Rather than exporting HAL pins and functions, these components provide the forward and inverse kinematics definitions for LinuxCNC.

trivkins − Trivial Kinematics

There is a 1:1 correspondence between joints and axes. Most standard milling machines and lathes use the trivial kinematics module.

rotatekins − Rotated Kinematics

The X and Y axes are rotated 45 degrees compared to the joints 0 and 1.

tripodkins − Tripod Kinematics

The joints represent the distance of the controlled point from three predefined locations (the motors), giving three degrees of freedom in position (XYZ)

tripodkins.Bx
tripodkins.Cx
tripodkins.Cy
The location of the three motors is (0,0), (Bx,0), and (Cx,Cy)

genhexkins − Hexapod Kinematics

Gives six degrees of freedom in position and orientation (XYZABC). The location of the motors is defined at compile time.

maxkins − 5-axis kinematics example

Kinematics for Chris Radek's tabletop 5 axis mill named 'max' with tilting head (B axis) and horizintal rotary mounted to the table (C axis). Provides UVW motion in the rotated coordinate system. The source file, maxkins.c, may be a useful starting point for other 5-axis systems.

genserkins − generalized serial kinematics

Kinematics that can model a general serial-link manipulator with up to 6 angular joints.

The kinematics use Denavit-Hartenberg definition for the joint and links. The DH definitions are the ones used by John J Craig in "Introduction to Robotics: Mechanics and Control" The parameters for the manipulator are defined by hal pins.

genserkins.A-*N*
genserkins.ALPHA-*N*
genserkins.D-*N*
Parameters describing the *N*th joint's geometry.

pumakins − kinematics for puma typed robots

Kinematics for a puma-style robot with 6 joints

pumakins.A2
pumakins.A3
pumakins.D3
pumakins.D4
Describe the geometry of the robot

scarakins − kinematics for SCARA-type robots

scarakins.D1
Vertical distance from the ground plane to the center of the inner arm.

scarakins.D2
Horizontal distance between joint[0] axis and joint[1] axis, ie. the length of the inner arm.

scarakins.D3
Vertical distance from the center of the inner arm to the center of the outer arm. May be positive or negative depending on the structure of the robot.

scarakins.D4
Horizontal distance between joint[1] axis and joint[2] axis, ie. the length of the outer arm.

scarakins.D5
Vertical distance from the end effector to the tooltip. Positive means the tooltip is lower than the end effector, and is the normal case.

scarakins.D6
Horizontal distance from the centerline of the end effector (and the joints 2 and 3 axis) and the tooltip. Zero means the tooltip is on the centerline. Non-zero values should be positive, if negative they introduce a 180 degree offset on the value of joint[3].

SEE ALSO

Kinematics section in the LinuxCNC documentation

NAME

knob2float − Convert counts (probably from an encoder) to a float value

SYNOPSIS

loadrt knob2float [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

knob2float.*N* (requires a floating-point thread)

PINS

knob2float.*N***.counts** s32 in
Counts

knob2float.*N***.enable** bit in
When TRUE, output is controlled by count, when FALSE, output is fixed

knob2float.*N***.scale** float in
Amount of output change per count

knob2float.*N***.out** float out
Output value

PARAMETERS

knob2float.*N***.max-out** float rw (default: *1.0*)
Maximum output value, further increases in count will be ignored

knob2float.*N***.min-out** float rw (default: *0.0*)
Minimum output value, further decreases in count will be ignored

LICENSE

GPL

NAME

latencybins − comp utility for scripts/latencyhistogram

SYNOPSIS

Usage:
Read availablebins pin for the number of bins available.
Set the maxbinnumber pin for the number of +/- bins.
Ensure maxbinnumber <= availablebins
For maxbinnumber = N, the bins are numbered:
-N ... 0 ... + N bins
(the -0 bin is not populated)
(total effective bins = 2*maxbinnumber +1)
Set nsbinsize pin for the binsize (ns)
Iterate:
Set index pin to a bin number: 0 <= index <= maxbinnumber.
Read check pin and verify that check pin == index pin.
Read pbinvalue,nbinvalue,pextra,nextra pins.
(pbinvalue is count for bin = +index)
(nbinvalue is count for bin = -index)
(pextra is count for all bins > maxbinnumber)
(nextra is count for all bins < maxbinnumber)

If index is out of range (index < 0 or index > maxbinnumber)
then pbinvalue = nbinvalue = -1.
The reset pin may be used to restart.
The latency pin outputs the instantaneous latency.

Maintainers note: hardcoded for MAXBINNUMBER==1000

FUNCTIONS

latencybins.*N*

PINS

latencybins.*N***.maxbinnumber** s32 in (default: *1000*)
latencybins.*N***.index** s32 in
latencybins.*N***.reset** bit in
latencybins.*N***.nsbinsize** s32 in
latencybins.*N***.check** s32 out
latencybins.*N***.latency** s32 out
latencybins.*N***.pbinvalue** s32 out
latencybins.*N***.nbinvalue** s32 out
latencybins.*N***.pextra** s32 out
latencybins.*N***.nextra** s32 out
latencybins.*N***.availablebins** s32 out (default: *1000*)

LICENSE

GPL

NAME

lcd − Stream HAL data to an LCD screen

SYNOPSIS

loadrt lcd fmt_strings=""Plain Text %4.4f\nAnd So on|Second Page, Next Inst""

FUNCTIONS

lcd (requires a floating-point thread). All LCD instances are updated by the same function.

PINS

lcd.*NN*.out (u32) out

The output byte stream is sent via this pin. One character is sent every thread invocation. There in no handshaking provided.

lcd.*NN*.page.*PP*.arg.*NN* (float/s32/u32/bit) in

The input pins have types matched to the format string specifiers.

lcd.*NN*.page_num (u32) in

Selects the page number. Multiple layouts may be defined, and this pin switches between them.

lcd.*NN*.contrast (float) in

Attempts to set the contrast of the LCD screen using the byte sequence ESC C and then a value from 0x20 to 0xBF. (matching the Mesa 7i73). The value should be between 0 and 1.

PARAMETERS

lcd.*NN*.decimal-separator (u32) rw

Sets the decimal separator used for floating point numbers. The default value is 46 (0x2E) which corresponds to ".". If a comma is required then set this parameter to 44 (0x2C).

DESCRIPTION

lcd takes format strings much like those used in C and many other languages in the printf and scanf functions and their variants.

The component was written specifically to support the Mesa 7i73 pendant controller, however it may be of use streaming data to other character devices and, as the output format mimics the ADM3 terminal format, it could be used to stream data to a serial device. Perhaps even a genuine ADM3. The strings contain a mixture of text values which are displayed directly, "escaped" formatting codes and numerical format descriptors. For a detailed description of formatting codes see: http://en.wikipedia.org/wiki/Printf

The component can be configured to display an unlimited number of differently-formatted pages, which may be selected with a HAL pin.

Escaped codes

\n Inserts a clear-to-end, carriage return and line feed character. This will still linefeed and clear even if an automatic wrap has occurred (lcd has no knowledge of the width of the lcd display.) To print in the rightmost column it is necessary to allow the format to wrap and omit the \n code.

\t Inserts a tab (actually 4 spaces in the current version rather than a true tab.)

\NN inserts the character defined by the hexadecimal code NN.

\\ Inserts a literal \.

Numerical formats

lcd differs slightly from the standard printf conventions. One significant difference is that width

limits are strictly enforced to prevent the LCD display wrapping and spoiling the layout. The field width includes the sign character so that negative numbers will often have a smaller valid range than positive. Numbers that do not fit in the specified width are displayed as a line of asterisks (**********).

Each format begins with a "%" symbol. (For a literal % use "%%"). Immediately after the % the following modifiers may be used:

" " (space) Pad the number to the specified width with spaces. This is the default and is not strictly necessary.

"0" Pad the number to the specified width with the numeral 0.

"+" Force display of a + symbol before positive numbers. This (like the - sign) will appear immediately to the left of the digits for a space-padded number and in the extreme left position for a 0-padded number.

"1234567890" A numerical entry (other than the leading 0 above) defines the total number of characters to display including the decimal separator and the sign. Whilst this number can be as many digits as required the maximum field width is 20 characters. The inherent precison of the "double" data type means that more than 14 digits will tend to show errors in the least significant digits. The integer data types will never fill more than 10 decimal digits.

Following the width specifier should be the decimal specifier. This can only be a full-stop character (.) as the comma (,) is used as the instance separator. Currently lcd does not access the locale information to determine the correct separator and the **decimal-separator** parameter should be used.

Following the decimal separator should be a number that determines how many places of decimals to display. This entry is ignored in the case of integer formats.

All the above modifiers are optional, but to specify a decimal precision the decimal point must precede the precision. For example %.3f.
The default decimal precision is 4.

The numerical formats supported are:

%f %F (for example, %+09.3f) These create a floating-point type HAL pin. The example would be displayed in a 9-character field, with 3 places of decimals, . as a decimal separator, padded to the left with 0s and with a sign displayed for both positive and negative. Conversely a plain %f would be 6 digits of decimal, variable format width, with a sign only shown for negative numbers. both %f and %F create exactly the same format.

%i %d (For example %+ 4d) Creates a signed (s32) HAL pin. The example would display the value at a fixed 4 characters, space padded, width including the + giving a range of +999 to -999. %i and %d create identical output.

%u (for example %08u) Creates an unsigned (u32) HAL pin. The example would be a fixed 8 characters wide, padded with zeros.

%x, %X Creates an unsigned (u32) HAL pin and displays the value in Hexadecimal. Both %x and %X display capital letters for digits ABCDEF. A width may be specified, though the u32 HAL type is only 8 hex digits wide.

%o Creates an unsigned (u32) pin and displays the value in Octal.

%c Creates a u32 HAL pin and displays the character corresponding to the value of the pin. Values less than 32 (space) are suppressed. A width specifier may be used, for example %20c might be used to create a complete line of one character.

%b This specifier has no equivalent in printf. It creates a bit (boolean) type HAL pin. The b should be followed by two characters and the display will show the first of these when the pin is true, and the second when false. Note that the characters follow, not preceed the "b", unlike the case with other formats. The characters may be "escaped" Hex values. For example "%b\FF " will display a solid black block if true, and a space if false and "%b\7F\7E" would display right-arrow for false and left-arrow for true. An unexpected value of 'E' indicates a formatting error.

Pages The page separator is the "|" (pipe) character. (if the actual character is needed then \7C may be used). A "Page" in this context refers to a separate format which may be displayed on the same display.

Instances The instance separator is the comma. This creates a completely separate lcd instance, for example to drive a second lcd display on the second 7i73. The use of comma to separate instances is built in to the modparam reading code so not even escaped commas "\," can be used. A comma may be displayed by using the \2C sequence.

AUTHOR

Andy Pugh

LICENSE

GPL

NAME

limit1 − Limit the output signal to fall between min and max

SYNOPSIS

loadrt limit1 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

limit1.*N* (requires a floating-point thread)

PINS

limit1.*N***.in** float in
limit1.*N***.out** float out

PARAMETERS

limit1.*N***.min** float rw (default: *-1e20*)
limit1.*N***.max** float rw (default: *1e20*)

LICENSE

GPL

NAME

limit2 − Limit the output signal to fall between min and max and limit its slew rate to less than maxv per second. When the signal is a position, this means that position and velocity are limited.

SYNOPSIS

loadrt limit2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

limit2.*N* (requires a floating-point thread)

PINS

limit2.*N***.in** float in
limit2.*N***.out** float out
limit2.*N***.load** bit in
When TRUE, immediately set **out to in**, ignoring maxv

PARAMETERS

limit2.*N***.min** float rw (default: *-1e20*)
limit2.*N***.max** float rw (default: *1e20*)
limit2.*N***.maxv** float rw (default: *1e20*)

LICENSE

GPL

NAME

limit3 − Limit the output signal to fall between min and max, limit its slew rate to less than maxv per second, and limit its second derivative to less than maxa per second squared. When the signal is a position, this means that the position, velocity, and acceleration are limited.

SYNOPSIS

loadrt limit3 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

limit3.*N* (requires a floating-point thread)

PINS

limit3.*N***.in** float in
limit3.*N***.out** float out
limit3.*N***.load** bit in
When TRUE, immediately set **out to in**, ignoring maxv and maxa

limit3.*N***.min** float in (default: *-1e20*)
limit3.*N***.max** float in (default: *1e20*)
limit3.*N***.maxv** float in (default: *1e20*)
limit3.*N***.maxa** float in (default: *1e20*)

LICENSE

GPL

NAME

logic − LinuxCNC HAL component providing configurable logic functions

SYNOPSIS

loadrt logic [count=*N***|names=***name1***[,***name2...***]] [personality=***P,P,...***]**

DESCRIPTION

General ‘logic function’ component. Can perform ‘and’, ‘or’ and ‘xor’ of up to 16 inputs.

Determine the proper value for ‘personality’ by adding the inputs and outputs then convert to hex:

- The number of input pins, usually from 2 to 16
- 256 (0x100) if the ‘and’ output is desired
- 512 (0x200) if the ‘or’ output is desired
- 1024 (0x400) if the ‘xor’ (exclusive or) output is desired

Outputs can be combined, for example 2 + 256 + 1024 = 1282 converted to hex would be 0x502 and would have two inputs and have both ‘xor’ and ‘and’ outputs.

FUNCTIONS

logic.*N*

PINS

logic.*N***.in-***MM* bit in (MM=00..personality & 0xff)
logic.*N***.and** bit out [if personality & 0x100]
logic.*N***.or** bit out [if personality & 0x200]
logic.*N***.xor** bit out [if personality & 0x400]

LICENSE

GPL

NAME

lowpass − Low-pass filter

SYNOPSIS

loadrt lowpass [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

lowpass.*N* (requires a floating-point thread)

PINS

lowpass.*N***.in** float in

lowpass.*N***.out** float out

out += (in - out) * gain

lowpass.*N***.load** bit in

When TRUE, copy **in** to **out** instead of applying the filter equation.

PARAMETERS

lowpass.*N***.gain** float rw

NOTES

The effect of a specific **gain** value is dependent on the period of the function that **lowpass.***N* is added to

LICENSE

GPL

NAME

lut5 − Arbitrary 5-input logic function based on a look-up table

SYNOPSIS

loadrt lut5 [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

lut5 constructs a logic function with up to 5 inputs using a **l**ook-**u**p **t**able. The value for **function** can be determined by writing the truth table, and computing the sum of **all** the **weights** for which the output value would be TRUE. The weights are hexadecimal not decimal so hexadecimal math must be used to sum the weights. A wiki page has a calculator to assist in computing the proper value for function.

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Lut5

Note that LUT5 will generate any of the 4,294,967,296 logical functions of 5 inputs so **AND**, **OR**, **NAND**, **NOR**, **XOR** and every other combinatorial function is possible.

Example Functions

A 5-input *and* function is TRUE only when all the inputs are true, so the correct value for **function** is **0x80000000**.

A 2-input *or* function would be the sum of **0x2** + **0x4** + **0x8**, so the correct value for **function** is **0xe**.

A 5-input *or* function is TRUE whenever any of the inputs are true, so the correct value for **function** is **0xfffffffe**. Because every weight except **0x1** is true the function is the sum of every line except the first one.

A 2-input *xor* function is TRUE whenever exactly one of the inputs is true, so the correct value for **function** is **0x6**. Only **in-0** and **in-1** should be connected to signals, because if any other bit is **TRUE** then the output will be **FALSE**.

Weights for each line of truth table					
Bit 4	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Weight**
0	0	0	0	0	0x1
0	0	0	0	1	0x2
0	0	0	1	0	0x4
0	0	0	1	1	0x8
0	0	1	0	0	0x10
0	0	1	0	1	0x20
0	0	1	1	0	0x40
0	0	1	1	1	0x80
0	1	0	0	0	0x100
0	1	0	0	1	0x200
0	1	0	1	0	0x400
0	1	0	1	1	0x800
0	1	1	0	0	0x1000
0	1	1	0	1	0x2000
0	1	1	1	0	0x4000
0	1	1	1	1	0x8000
1	0	0	0	0	0x10000
1	0	0	0	1	0x20000
1	0	0	1	0	0x40000
1	0	0	1	1	0x80000
1	0	1	0	0	0x100000
1	0	1	0	1	0x200000
1	0	1	1	0	0x400000
1	0	1	1	1	0x800000
1	1	0	0	0	0x1000000
1	1	0	0	1	0x2000000
1	1	0	1	0	0x4000000
1	1	0	1	1	0x8000000
1	1	1	0	0	0x10000000
1	1	1	0	1	0x20000000
1	1	1	1	0	0x40000000
1	1	1	1	1	0x80000000

FUNCTIONS

lut5.*N*

PINS

lut5.*N***.in-0** bit in
lut5.*N***.in-1** bit in
lut5.*N***.in-2** bit in
lut5.*N***.in-3** bit in
lut5.*N***.in-4** bit in
lut5.*N***.out** bit out

PARAMETERS

lut5.*N***.function** u32 rw

LICENSE

GPL

NAME

maj3 − Compute the majority of 3 inputs

SYNOPSIS

loadrt maj3 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

maj3.*N*

PINS

maj3.*N***.in1** bit in
maj3.*N***.in2** bit in
maj3.*N***.in3** bit in
maj3.*N***.out** bit out

PARAMETERS

maj3.*N***.invert** bit rw

LICENSE

GPL

NAME

match8 − 8-bit binary match detector

SYNOPSIS

loadrt match8 [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

match8.*N*

PINS

match8.*N***.in** bit in (default: *TRUE*)
cascade input - if false, output is false regardless of other inputs

match8.*N***.a0** bit in
match8.*N***.a1** bit in
match8.*N***.a2** bit in
match8.*N***.a3** bit in
match8.*N***.a4** bit in
match8.*N***.a5** bit in
match8.*N***.a6** bit in
match8.*N***.a7** bit in
match8.*N***.b0** bit in
match8.*N***.b1** bit in
match8.*N***.b2** bit in
match8.*N***.b3** bit in
match8.*N***.b4** bit in
match8.*N***.b5** bit in
match8.*N***.b6** bit in
match8.*N***.b7** bit in
match8.*N***.out** bit out
true only if in is true and a[m] matches b[m] for m = 0 thru 7

LICENSE

GPL

NAME

message − Display a message

SYNOPSIS

loadrt message [count=*N***|names=***name1***[,***name2...***]] [messages=***N***]**

messages

The messages to display. These should be listed, comma-delimited, inside a single set of quotes. See the "Description" section for an example. If there are more messages than "count" or "names" then the excess will be ignored. If there are fewer messages than "count" or "names" then an error will be raised and the component will not load.

DESCRIPTION

Allows HAL pins to trigger a message. Example hal commands:

```
loadrt message names=oillow,oilpressure,inverterfail messages="Slideway oil low,No oil pressure,Spindle inverter fault"
addf oillow servo-thread
addf oilpressure servo-thread
addf inverterfail servo-thread

setp oillow.edge 0 #this pin should be active low
net no-oil classicladder.0.out-21 oillow.trigger
net no-pressure classicladder.0.out-22 oilpressure.trigger
net no-inverter classicladder.0.out-23 inverterfail.trigger
```

When any pin goes active, the corresponding message will be displayed.

FUNCTIONS

message.*N*

Display a message

PINS

message.*N***.trigger** bit in (default: *FALSE*)

signal that triggers the message

message.*N***.force** bit in (default: *FALSE*)

A FALSE->TRUE transition forces the message to be displayed again if the trigger is active

PARAMETERS

message.*N***.edge** bit rw (default: *TRUE*)

Selects the desired edge: TRUE means falling, FALSE means rising

LICENSE

GPL v2

NAME

minmax − Track the minimum and maximum values of the input to the outputs

SYNOPSIS

loadrt minmax [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

minmax.*N* (requires a floating-point thread)

PINS

minmax.*N***.in** float in
minmax.*N***.reset** bit in
When reset is asserted, 'in' is copied to the outputs

minmax.*N***.max** float out
minmax.*N***.min** float out

LICENSE

GPL

NAME

motion − accepts NML motion commands, interacts with HAL in realtime

SYNOPSIS

loadrt motmod [base_period_nsec=*period***] [base_thread_fp=***0 or 1***] [servo_period_nsec=***period***] [traj_period_nsec=***period***] [num_joints=***[0-9]***] ([num_dio=***[1-64]***] [num_aio=***[1-16]***])**

DESCRIPTION

By default, the base thread does not support floating point. Software stepping, software encoder counting, and software pwm do not use floating point. **base_thread_fp** can be used to enable floating point in the base thread (for example for brushless DC motor control).

These pins and parameters are created by the realtime **motmod** module. This module provides a HAL interface for LinuxCNC's motion planner. Basically **motmod** takes in a list of waypoints and generates a nice blended and constraint-limited stream of joint positions to be fed to the motor drives.

Optionally the number of Digital I/O is set with num_dio. The number of Analog I/O is set with num_aio. The default is 4 each.

Pin names starting with "**axis**" are actually joint values, but the pins and parameters are still called "**axis.***N*". They are read and updated by the motion-controller function.

PINS

axis.*N***.amp-enable-out** OUT BIT
TRUE if the amplifier for this joint should be enabled

axis.*N***.amp-fault-in** IN BIT
Should be driven TRUE if an external fault is detected with the amplifier for this joint

axis.*N***.home-sw-in** IN BIT
Should be driven TRUE if the home switch for this joint is closed

axis.*N***.homing** OUT BIT
TRUE if the joint is currently homing

axis.*N***.index-enable** IO BIT
Should be attached to the index-enable pin of the joint's encoder to enable homing to index pulse

axis.*N***.is-unlocked** IN BIT
If the axis is a locked rotary the unlocked sensor should be connected to this pin

axis.*N***.jog-counts** IN S32
Connect to the "counts" pin of an external encoder to use a physical jog wheel.

axis.*N***.jog-enable** IN BIT
When TRUE (and in manual mode), any change to "jog-counts" will result in motion. When false, "jog-counts" is ignored.

axis.*N*.jog-scale IN FLOAT
Sets the distance moved for each count on "jog-counts", in machine units.

axis.*N*.jog-vel-mode IN BIT
When FALSE (the default), the jogwheel operates in position mode. The axis will move exactly jog-scale units for each count, regardless of how long that might take. When TRUE, the wheel operates in velocity mode - motion stops when the wheel stops, even if that means the commanded motion is not completed.

axis.*N*.joint-pos-cmd OUT FLOAT
The joint (as opposed to motor) commanded position. There may be several offsets between the joint and motor coordinates: backlash compensation, screw error compensation, and home offsets.

axis.*N*.joint-pos-fb OUT FLOAT
The joint feedback position. This value is computed from the actual motor position minus joint offsets. Useful for machine visualization.

axis.*N*.motor-pos-cmd OUT FLOAT
The commanded position for this joint.

axis.*N*.motor-pos-fb IN FLOAT
The actual position for this joint.

axis.*N*.neg-lim-sw-in IN BIT
Should be driven TRUE if the negative limit switch for this joint is tripped.

axis.*N*.pos-lim-sw-in IN BIT
Should be driven TRUE if the positive limit switch for this joint is tripped.

axis.*N*.unlock OUT BIT
TRUE if the axis is a locked rotary and a move is commanded.

motion.adaptive-feed IN FLOAT
When adaptive feed is enabled with M52 P1, the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and motion.feed-hold.

motion.analog-in-*NN* IN FLOAT
These pins are used by M66 Enn wait-for-input mode.

motion.analog-out-*NN* OUT FLOAT
These pins are used by M67-68.

motion.coord-error OUT BIT
TRUE when motion has encountered an error, such as exceeding a soft limit

motion.coord-mode OUT BIT
TRUE when motion is in "coordinated mode", as opposed to "teleop mode"

motion.current-vel OUT FLOAT
Current cartesian velocity

motion.digital-in-*NN* IN BIT
These pins are used by M66 Pnn wait-for-input mode.

motion.digital-out-*NN* OUT BIT
These pins are controlled by the M62 through M65 words.

motion.distance-to-go OUT FLOAT
Distance remaining in the current move

motion.enable IN BIT
If this bit is driven FALSE, motion stops, the machine is placed in the "machine off" state, and a message is displayed for the operator. For normal motion, drive this bit TRUE.

motion.feed-hold IN BIT
When Feed Stop Control is enabled with M53 P1, and this bit is TRUE, the feed rate is set to 0.

motion.motion-inpos OUT BIT
TRUE if the machine is in position.

motion.probe-input IN BIT
G38.x uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.

motion.program-line OUT S32

motion.requested-vel OUT FLOAT
The requested velocity with no adjustments for feed override

motion.spindle-at-speed IN BIT
Motion will pause until this pin is TRUE, under the following conditions: before the first feed move after each spindle start or speed change; before the start of every chain of spindle-synchronized moves; and if in CSS mode, at every rapid->feed transition.

motion.spindle-brake OUT BIT
TRUE when the spindle brake should be applied

motion.spindle-forward OUT BIT
TRUE when the spindle should rotate forward

motion.spindle-index-enable I/O BIT
For correct operation of spindle synchronized moves, this signal must be hooked to the index-enable pin of the spindle encoder.

motion.spindle-on OUT BIT
TRUE when spindle should rotate

motion.spindle-reverse OUT BIT
TRUE when the spindle should rotate backward

motion.spindle-revs IN FLOAT
For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder.

motion.spindle-speed-in IN FLOAT
Actual spindle speed feedback in revolutions per second; used for G96 feed-per-revolution and constant surface speed modes.

motion.spindle-speed-out OUT FLOAT
Desired spindle speed in rotations per minute

motion.spindle-speed-out-rps OUT float
Desired spindle speed in rotations per second

motion.tooloffset.x OUT FLOAT

motion.tooloffset.y OUT FLOAT

motion.tooloffset.z OUT FLOAT

motion.tooloffset.a OUT FLOAT

motion.tooloffset.b OUT FLOAT

motion.tooloffset.c OUT FLOAT

motion.tooloffset.u OUT FLOAT

motion.tooloffset.v OUT FLOAT

motion.tooloffset.w OUT FLOAT
Current tool offset in all 9 axes.

DEBUGGING PINS

Many of the pins below serve as debugging aids, and are subject to change or removal at any time.

axis.*N*.active OUT BIT
TRUE when this joint is active

axis.*N*.backlash-corr OUT FLOAT
Backlash or screw compensation raw value

axis.*N*.backlash-filt OUT FLOAT
Backlash or screw compensation filtered value (respecting motion limits)

axis.*N*.backlash-vel OUT FLOAT
Backlash or screw compensation velocity

axis.*N*.coarse-pos-cmd OUT FLOAT

axis.*N*.error OUT BIT
TRUE when this joint has encountered an error, such as a limit switch closing

axis.*N*.f-error OUT FLOAT
The actual following error

axis.*N*.f-error-lim OUT FLOAT
The following error limit

axis.*N*.f-errored OUT BIT
TRUE when this joint has exceeded the following error limit

axis.*N*.faulted OUT BIT

axis.*N*.free-pos-cmd OUT FLOAT
The "free planner" commanded position for this joint.

axis.*N*.free-tp-enable OUT BIT
TRUE when the "free planner" is enabled for this joint

axis.*N*.free-vel-lim OUT FLOAT
The velocity limit for the free planner

axis.*N*.homed OUT BIT
TRUE if the joint has been homed

axis.*N*.in-position OUT BIT
TRUE if the joint is using the "free planner" and has come to a stop

axis.*N*.joint-vel-cmd OUT FLOAT
The joint’s commanded velocity

axis.*N*.kb-jog-active OUT BIT

axis.*N*.neg-hard-limit OUT BIT
The negative hard limit for the joint

axis.*N*.pos-hard-limit OUT BIT
The positive hard limit for the joint

axis.*N*.wheel-jog-active OUT BIT

motion.in-position OUT BIT
Same as the pin motion.motion-inpos

motion.motion-enabled IN BIT

motion.on-soft-limit OUT BIT

motion.program-line OUT S32

motion.teleop-mode OUT BIT
TRUE when motion is in "teleop mode", as opposed to "coordinated mode"

PARAMETERS

Many of the parameters serve as debugging aids, and are subject to change or removal at any time.

motion-command-handler.time

motion-command-handler.tmax

motion-controller.time

motion-controller.tmax
Show information about the execution time of these HAL functions in CPU cycles

motion.debug-*
These values are used for debugging purposes.

motion.servo.last-period
The number of CPU cycles between invocations of the servo thread. Typically, this number divided by the CPU speed gives the time in seconds, and can be used to determine whether the realtime motion controller is meeting its timing constraints

motion.servo.overruns
By noting large differences between successive values of motion.servo.last-period, the motion controller can determine that there has probably been a failure to meet its timing constraints. Each time such a failure is detected, this value is incremented.

FUNCTIONS

Generally, these functions are both added to the servo-thread in the order shown.

motion-command-handler
Processes motion commands coming from user space

motion-controller
Runs the LinuxCNC motion controller

BUGS

This manual page is horribly incomplete.

SEE ALSO

iocontrol(1)

NAME

mult2 − Product of two inputs

SYNOPSIS

loadrt mult2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

mult2.*N* (requires a floating-point thread)

PINS

mult2.*N***.in0** float in
mult2.*N***.in1** float in
mult2.*N***.out** float out
out = in0 * in1

LICENSE

GPL

NAME

multiclick − Single-, double-, triple-, and quadruple-click detector

SYNOPSIS

loadrt multiclick [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

A click is defined as a rising edge on the 'in' pin, followed by the 'in' pin being True for at most 'max-hold-ns' nanoseconds, followed by a falling edge.

A double-click is defined as two clicks, separated by at most 'max-space-ns' nanoseconds with the 'in' pin in the False state.

I bet you can guess the definition of triple- and quadruple-click.

You probably want to run the input signal through a debounce component before feeding it to the multiclick detector, if the input is at all noisy.

The '*-click' pins go high as soon as the input detects the correct number of clicks.

The '*-click-only' pins go high a short while after the click, after the click separator space timeout has expired to show that no further click is coming. This is useful for triggering halui MDI commands.

FUNCTIONS

multiclick.*N*

Detect single-, double-, triple-, and quadruple-clicks

PINS

multiclick.*N***.in** bit in

The input line, this is where we look for clicks.

multiclick.*N***.single-click** bit out

Goes high briefly when a single-click is detected on the 'in' pin.

multiclick.*N***.single-click-only** bit out

Goes high briefly when a single-click is detected on the 'in' pin and no second click followed it.

multiclick.*N***.double-click** bit out

Goes high briefly when a double-click is detected on the 'in' pin.

multiclick.*N***.double-click-only** bit out

Goes high briefly when a double-click is detected on the 'in' pin and no third click followed it.

multiclick.*N***.triple-click** bit out

Goes high briefly when a triple-click is detected on the 'in' pin.

multiclick.*N***.triple-click-only** bit out

Goes high briefly when a triple-click is detected on the 'in' pin and no fourth click followed it.

multiclick.*N***.quadruple-click** bit out

Goes high briefly when a quadruple-click is detected on the 'in' pin.

multiclick.*N***.quadruple-click-only** bit out

Goes high briefly when a quadruple-click is detected on the 'in' pin and no fifth click followed it.

multiclick.*N***.state** s32 out

PARAMETERS

multiclick.*N***.invert-input** bit rw (default: *FALSE*)

If FALSE (the default), clicks start with rising edges. If TRUE, clicks start with falling edges.

multiclick.*N*.max-hold-ns u32 rw (default: *250000000*)
If the input is held down longer than this, it's not part of a multi-click. (Default 250,000,000 ns, 250 ms.)

multiclick.*N*.max-space-ns u32 rw (default: *250000000*)
If the input is released longer than this, it's not part of a multi-click. (Default 250,000,000 ns, 250 ms.)

multiclick.*N*.output-hold-ns u32 rw (default: *100000000*)
Positive pulses on the output pins last this long. (Default 100,000,000 ns, 100 ms.)

LICENSE

GPL

NAME

multiswitch − This component toggles between a specified number of output bits

SYNOPSIS

loadrt multiswitch personality=*P* **[cfg=***N***]**

cfg cfg should be a comma-separated list of sizes for example cfg=2,4,6 would create 3 instances of 2, 4 and 6 bits respectively.
Ignore the "personality" parameter, that is auto-generated

FUNCTIONS

multiswitch.*N* (requires a floating-point thread)

PINS

multiswitch.*N***.up** bit in (default: *false*)
Receives signal to toggle up

multiswitch.*N***.down** bit in (default: *false*)
Receives signal to toggle down

multiswitch.*N***.bit-***MM* bit out (MM=00..personality) (default: *false*)
Output bits

PARAMETERS

multiswitch.*N***.top-position** u32 rw
Number of positions

multiswitch.*N***.position** s32 rw
Current state (may be set in the HAL)

AUTHOR

ArcEye schooner30@tiscali.co.uk / Andy Pugh andy@bodgesoc.org

LICENSE

GPL

NAME

mux16 − Select from one of sixteen input values

SYNOPSIS

loadrt mux16 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

mux16.*N* (requires a floating-point thread)

PINS

mux16.*N***.use-graycode** bit in

This signifies the input will use Gray code instead of binary. Gray code is a good choice when using physical switches because for each increment only one select input changes at a time.

mux16.*N***.suppress-no-input** bit in

This suppresses changing the output if all select lines are false. This stops unwanted jumps in output between transitions of input. but make in00 unavaliable.

mux16.*N***.debounce-time** float in

sets debouce time in seconds. eg. .10 = a tenth of a second input must be stable this long before outputs changes. This helps to ignore 'noisy' switches.

mux16.*N***.sel***M* bit in (M=0..3)

Together, these determine which **in***N* value is copied to **out**.

mux16.*N***.out-f** float out

mux16.*N***.out-s** s32 out

Follows the value of one of the **in***N* values according to the four **sel** values and whether use-graycode is active. The s32 value will be trunuated and limited to the max and min values of signed values.

sel3=FALSE, **sel2=FALSE**, **sel1=FALSE**, **sel0=FALSE**

out follows **in0**

sel3=FALSE, **sel2=FALSE**, **sel1=FALSE**, **sel0=TRUE**

out follows **in1**

etc.

mux16.*N***.in***MM* float in (MM=00..15)

array of selectable outputs

PARAMETERS

mux16.*N***.elapsed** float r

Current value of the internal debounce timer for debugging.

mux16.*N***.selected** s32 r

Current value of the internal selection variable after conversion for debugging

LICENSE

GPL

NAME

mux2 − Select from one of two input values

SYNOPSIS

loadrt mux2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

mux2.*N* (requires a floating-point thread)

PINS

mux2.*N***.sel** bit in
mux2.*N***.out** float out
Follows the value of in0 if sel is FALSE, or in1 if sel is TRUE

mux2.*N***.in1** float in
mux2.*N***.in0** float in

LICENSE

GPL

NAME

mux4 − Select from one of four input values

SYNOPSIS

loadrt mux4 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

mux4.*N* (requires a floating-point thread)

PINS

mux4.*N***.sel0** bit in
mux4.*N***.sel1** bit in
Together, these determine which **in***N* value is copied to **out**.

mux4.*N***.out** float out
Follows the value of one of the **in***N* values according to the two **sel** values

sel1=FALSE, **sel0=FALSE**
out follows **in0**

sel1=FALSE, **sel0=TRUE**
out follows **in1**

sel1=TRUE, **sel0=FALSE**
out follows **in2**

sel1=TRUE, **sel0=TRUE**
out follows **in3**

mux4.*N***.in0** float in
mux4.*N***.in1** float in
mux4.*N***.in2** float in
mux4.*N***.in3** float in

LICENSE

GPL

NAME

mux8 − Select from one of eight input values

SYNOPSIS

loadrt mux8 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

mux8.*N* (requires a floating-point thread)

PINS

mux8.*N***.sel0** bit in
mux8.*N***.sel1** bit in
mux8.*N***.sel2** bit in
Together, these determine which **in***N* value is copied to **out**.

mux8.*N***.out** float out
Follows the value of one of the **in***N* values according to the three **sel** values

sel2=FALSE, **sel1=FALSE**, **sel0=FALSE**
out follows **in0**

sel2=FALSE, **sel1=FALSE**, **sel0=TRUE**
out follows **in1**

sel2=FALSE, **sel1=TRUE**, **sel0=FALSE**
out follows **in2**

sel2=FALSE, **sel1=TRUE**, **sel0=TRUE**
out follows **in3**

sel2=TRUE, **sel1=FALSE**, **sel0=FALSE**
out follows **in4**

sel2=TRUE, **sel1=FALSE**, **sel0=TRUE**
out follows **in5**

sel2=TRUE, **sel1=TRUE**, **sel0=FALSE**
out follows **in6**

sel2=TRUE, **sel1=TRUE**, **sel0=TRUE**
out follows **in7**

mux8.*N***.in0** float in
mux8.*N***.in1** float in
mux8.*N***.in2** float in
mux8.*N***.in3** float in
mux8.*N***.in4** float in
mux8.*N***.in5** float in
mux8.*N***.in6** float in
mux8.*N***.in7** float in

LICENSE

GPL

NAME

near − Determine whether two values are roughly equal.

SYNOPSIS

loadrt near [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

near.*N* (requires a floating-point thread)

PINS

near.*N***.in1** float in
near.*N***.in2** float in
near.*N***.out** bit out

> **out** is true if **in1** and **in2** are within a factor of **scale** (i.e., for in1 positive, in1/scale <= in2 <= in1*scale), OR if their absolute difference is no greater than **difference** (i.e., |in1-in2| <= difference). **out** is false otherwise.

PARAMETERS

near.*N***.scale** float rw (default: *1*)
near.*N***.difference** float rw (default: *0*)

LICENSE

GPL

NAME

not − Inverter

SYNOPSIS

loadrt not [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

not.*N*

PINS

not.*N***.in** bit in
not.*N***.out** bit out

LICENSE

GPL

NAME

offset − Adds an offset to an input, and subtracts it from the feedback value

SYNOPSIS

loadrt offset [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

offset.*N*.update-output (requires a floating-point thread)
Updated the output value by adding the offset to the input

offset.*N*.update-feedback (requires a floating-point thread)
Update the feedback value by subtracting the offset from the feedback

PINS

offset.*N*.offset float in
The offset value

offset.*N*.in float in
The input value

offset.*N*.out float out
The output value

offset.*N*.fb-in float in
The feedback input value

offset.*N*.fb-out float out
The feedback output value

LICENSE

GPL

NAME

oneshot − one-shot pulse generator

SYNOPSIS

loadrt oneshot [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

creates a variable-length output pulse when the input changes state. This function needs to run in a thread which supports floating point (typically the servo thread). This means that the pulse length has to be a multiple of that thread period, typically 1mS. For a similar function that can run in the base thread, and which offers higher resolution, see "edge".

FUNCTIONS

oneshot.*N* (requires a floating-point thread)
Produce output pulses from input edges

PINS

oneshot.*N***.in** bit in
Trigger input

oneshot.*N***.out** bit out
Active high pulse

oneshot.*N***.out-not** bit out
Active low pulse

oneshot.*N***.width** float in (default: *0*)
Pulse width in seconds

oneshot.*N***.time-left** float out
Time left in current output pulse

PARAMETERS

oneshot.*N***.retriggerable** bit rw (default: *TRUE*)
Allow additional edges to extend pulse

oneshot.*N***.rising** bit rw (default: *TRUE*)
Trigger on rising edge

oneshot.*N***.falling** bit rw (default: *FALSE*)
Trigger on falling edge

LICENSE

GPL

NAME

opto_ac5 − Realtime driver for opto22 PCI-AC5 cards

SYNOPSIS

loadrt opto_ac5 [portconfig0=0x*N***] [portconfig1=0x***N***]**

DESCRIPTION

These pins and parameters are created by the realtime **opto_ac5** module. The portconfig0 and portconfig1 variables are used to configure the two ports of each card. The first 24 bits of a 32 bit number represent the 24 i/o points of each port. The lowest (rightmost) bit would be HAL pin 0 which is header connector pin 47. Then next bit to the left would be HAL pin 1, header connector pin 45 and so on, untill bit 24 would be HAL pin 23 , header connector pin 1. "1" bits represent output points. So channel 0..11 as inputs and 12..23 as outputs would be represented by (in binary) 111111111111000000000000 which is 0xfff000 in hexadecimal. That is the number you would use Eg. loadrt opto_ac5 portconfig0=0xfff000

If no portconfig variable is specified the default configuration is 12 inputs then 12 outputs.

Up to 4 boards are supported. Boards are numbered starting at 0.

Portnumber can be 0 or 1. Port 0 is closes to the card bracket.

PINS

opto_ac5.[*BOARDNUMBER***].port[***PORTNUMBER***].in-[***PINNUMBER***]** OUT bit
opto_ac5.[*BOARDNUMBER***].port[***PORTNUMBER***].in-[***PINNUMBER***]**-not OUT bit

Connect a hal bit signal to this pin to read an i/o point from the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 0 is position 0 in a opto22 relay rack and would be pin 47 on the 50 pin header connector. The **-not** pin is inverted so that LOW gives TRUE and HIGH gives FALSE.

opto_ac5.[*BOARDNUMBER***].port[***PORTNUMBER***].out-[***PINNUMBER***]** IN bit

Connect a hal bit signal to this pin to write to an i/o point of the card. The PINNUMBER represents the position in the relay rack.Eg. PINNUMBER 23 is position 23 in a opto22 relay rack and would be pin 1 on the 50 pin header connector.

opto_ac5.[*BOARDNUMBER***].led[***NUMBER***]** OUT bit

Turns one of the on board LEDS on/off. LEDS are numbered 0 to 3.

PARAMETERS

opto_ac5.[*BOARDNUMBER***].port[***PORTNUMBER***].out-[***PINNUMBER***]**-invert W bit

When TRUE, invert the meaning of the corresponding **-out** pin so that TRUE gives LOW and FALSE gives HIGH.

FUNCTIONS

opto_ac5.0.digital-read

Add this to a thread to read all the input points.

opto_ac5.0.digital-write

Add this to a thread to write all the output points and LEDS.

BUGS

All boards are loaded with the same port configurations as the first board.

SEE ALSO

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?OptoPciAc5

NAME

or2 − Two-input OR gate

SYNOPSIS

loadrt or2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

or2.*N*

PINS

or2.*N***.in0** bit in
or2.*N***.in1** bit in
or2.*N***.out** bit out

out is computed from the value of **in0** and **in1** according to the following rule:

in0=FALSE in1=FALSE
out=FALSE

Otherwise,
out=TRUE

LICENSE

GPL

NAME

pcl720 − Driver for the Advantech PCL 720 card.

SYNOPSIS

loadrt pcl720 [ioaddr=*N*]

ioaddr Base address of card. Separate each card base address with a comma but no space to load more than one card. eg loadrt pcl720 ioaddr=0x200,0x200. use 0xNNN to define addresses in Hex

DESCRIPTION

This driver supports the Advantech PCL720 ISA card. It might work with the PCI version too, but this is untested.

It creates hal pins corresonding to the digital inputs and outputs, but does not support the the counters/timers.

FUNCTIONS

pcl720.*N*.read

Reads each of the digital inputs and updates the HAL pins

pcl720.*N*.write

Writes the values of the output HAL pins to the digital IO

pcl720.*N*.reset

Waits for the length of time specified by the **reset-time** parameter and resets any pins for which the **reset** parameter has been set. This can be used to allow step generators to make a step every thread rather than every other thread. This function must be added to the thread after the "write" function.

Do not use this function if you do not wish to reset any pins.

the stepgen **step-space** parameter should be set to 0 to make use of this function.

PINS

pcl720.*N*.pin-*MM*-out bit in (MM=00..31)

Output pins

pcl720.*N*.pin-*MM*-in bit out (MM=00..31)

Input pins

pcl720.*N*.pin-*MM*-in-not bit out (MM=00..31)

Inverted version of each input pin

pcl720.*N*.wait-clocks u32 out

PARAMETERS

pcl720.*N*.reset-time u32 rw (default: *5000*)

The time in nanoseconds after the write function has run to reset the pins for which the "reset" parameter is set.

pcl720.*N*.pin-*MM*-reset bit rw (MM=00..31)

specifies if the pin should be reset by the "reset" function

pcl720.*N*.pin-*MM*-out-invert bit rw (MM=00..31)

Set to true to invert the sense of the output pin

AUTHOR

Andy Pugh

LICENSE

GPL

NAME

pid − proportional/integral/derivative controller

SYNOPSIS

loadrt pid [num_chan=*num* **| names=***name1***[,***name2*...**]] [debug=***dbg*]

DESCRIPTION

pid is a classic Proportional/Integral/Derivative controller, used to control position or speed feedback loops for servo motors and other closed-loop applications.

pid supports a maximum of sixteen controllers. The number that are actually loaded is set by the **num_chan** argument when the module is loaded. Alternatively, specify names= and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is three. If **debug** is set to 1 (the default is 0), some additional HAL parameters will be exported, which might be useful for tuning, but are otherwise unnecessary.

NAMING

The names for pins, parameters, and functions are prefixed as:
pid.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **pid.N.** format is shown in the following descriptions.

FUNCTIONS

pid.*N***.do-pid-calcs** (uses floating-point) Does the PID calculations for control loop *N*.

PINS

pid.*N***.command** float in
The desired (commanded) value for the control loop.

pid.*N***.command-deriv** float in
The derivative of the desired (commanded) value for the control loop. If no signal is connected then the derivative will be estimated numerically.

pid.*N***.feedback** float in
The actual (feedback) value, from some sensor such as an encoder.

pid.*N***.feedback-deriv** float in
The derivative of the actual (feedback) value for the control loop. If no signal is connected then the derivative will be estimated numerically. When the feedback is from a quantized position source (e.g., encoder feedback position), behavior of the D term can be improved by using a better velocity estimate here, such as the velocity output of encoder(9) or hostmot2(9).

pid.*N***.error-previous-target** bit in
Use previous invocation's target vs. current position for error calculation, like the motion controller expects. This may make torque-mode position loops and loops requiring a large I gain easier to tune, by eliminating velocity-dependent following error.

pid.*N***.error** float out
The difference between command and feedback.

pid.*N***.output** float out
The output of the PID loop, which goes to some actuator such as a motor.

pid.*N***.enable** bit in
When true, enables the PID calculations. When false, **output** is zero, and all internal integrators, etc, are reset.

pid.*N*.index-enable bit in

On the falling edge of **index-enable**, pid does not update the internal command derivative estimate. On systems which use the encoder index pulse, this pin should be connected to the index-enable signal. When this is not done, and FF1 is nonzero, a step change in the input command causes a single-cycle spike in the PID output. On systems which use exactly one of the **-deriv** inputs, this affects the D term as well.

pid.*N*.saturated bit out

When true, the current PID output is saturated. That is,

output = ± **maxoutput**.

pid.*N*.saturated-s float out

pid.*N*.saturated-count s32 out

When true, the output of PID was continually saturated for this many seconds (**saturated-s**) or periods (**saturated-count**).

PARAMETERS

pid.*N*.Pgain float rw

Proportional gain. Results in a contribution to the output that is the error multiplied by **Pgain**.

pid.*N*.Igain float rw

Integral gain. Results in a contribution to the output that is the integral of the error multiplied by **Igain**. For example an error of 0.02 that lasted 10 seconds would result in an integrated error (**errorI**) of 0.2, and if **Igain** is 20, the integral term would add 4.0 to the output.

pid.*N*.Dgain float rw

Derivative gain. Results in a contribution to the output that is the rate of change (derivative) of the error multiplied by **Dgain**. For example an error that changed from 0.02 to 0.03 over 0.2 seconds would result in an error derivative (**errorD**) of of 0.05, and if **Dgain** is 5, the derivative term would add 0.25 to the output.

pid.*N*.bias float rw

bias is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. **bias** is turned off when the PID loop is disabled, just like all other components of the output. If a non-zero output is needed even when the PID loop is disabled, it should be added with an external HAL sum2 block.

pid.*N*.FF0 float rw

Zero order feed-forward term. Produces a contribution to the output that is **FF0** multiplied by the commanded value. For position loops, it should usually be left at zero. For velocity loops, **FF0** can compensate for friction or motor counter-EMF and may permit better tuning if used properly.

pid.*N*.FF1 float rw

First order feed-forward term. Produces a contribution to the output that **FF1** multiplied by the derivative of the commanded value. For position loops, the contribution is proportional to speed, and can be used to compensate for friction or motor CEMF. For velocity loops, it is proportional to acceleration and can compensate for inertia. In both cases, it can result in better tuning if used properly.

pid.*N*.FF2 float rw

Second order feed-forward term. Produces a contribution to the output that is **FF2** multiplied by the second derivative of the commanded value. For position loops, the contribution is proportional to acceleration, and can be used to compensate for inertia. For velocity loops, it should usually be left at zero.

pid.*N*.deadband float rw

Defines a range of "acceptable" error. If the absolute value of **error** is less than **deadband**, it will be treated as if the error is zero. When using feedback devices such as encoders that are inherently quantized, the deadband should be set slightly more than one-half count, to prevent the control

loop from hunting back and forth if the command is between two adjacent encoder values. When the absolute value of the error is greater than the deadband, the deadband value is subtracted from the error before performing the loop calculations, to prevent a step in the transfer function at the edge of the deadband. (See **BUGS**.)

pid.*N*.maxoutput float rw
Output limit. The absolute value of the output will not be permitted to exceed **maxoutput**, unless **maxoutput** is zero. When the output is limited, the error integrator will hold instead of integrating, to prevent windup and overshoot.

pid.*N*.maxerror float rw
Limit on the internal error variable used for P, I, and D. Can be used to prevent high **Pgain** values from generating large outputs under conditions when the error is large (for example, when the command makes a step change). Not normally needed, but can be useful when tuning non-linear systems.

pid.*N*.maxerrorD float rw
Limit on the error derivative. The rate of change of error used by the **Dgain** term will be limited to this value, unless the value is zero. Can be used to limit the effect of **Dgain** and prevent large output spikes due to steps on the command and/or feedback. Not normally needed.

pid.*N*.maxerrorI float rw
Limit on error integrator. The error integrator used by the **Igain** term will be limited to this value, unless it is zero. Can be used to prevent integrator windup and the resulting overshoot during/after sustained errors. Not normally needed.

pid.*N*.maxcmdD float rw
Limit on command derivative. The command derivative used by **FF1** will be limited to this value, unless the value is zero. Can be used to prevent **FF1** from producing large output spikes if there is a step change on the command. Not normally needed.

pid.*N*.maxcmdDD float rw
Limit on command second derivative. The command second derivative used by **FF2** will be limited to this value, unless the value is zero. Can be used to prevent **FF2** from producing large output spikes if there is a step change on the command. Not normally needed.

pid.*N*.errorI float ro (only if debug=1)
Integral of error. This is the value that is multiplied by **Igain** to produce the Integral term of the output.

pid.*N*.errorD float ro (only if debug=1)
Derivative of error. This is the value that is multiplied by **Dgain** to produce the Derivative term of the output.

pid.*N*.commandD float ro (only if debug=1)
Derivative of command. This is the value that is multiplied by **FF1** to produce the first order feed-forward term of the output.

pid.*N*.commandDD float ro (only if debug=1)
Second derivative of command. This is the value that is multiplied by **FF2** to produce the second order feed-forward term of the output.

BUGS

Some people would argue that deadband should be implemented such that error is treated as zero if it is within the deadband, and be unmodified if it is outside the deadband. This was not done because it would cause a step in the transfer function equal to the size of the deadband. People who prefer that behavior are welcome to add a parameter that will change the behavior, or to write their own version of **pid**. However, the default behavior should not be changed.

Negative gains may lead to unwanted behavior. It is possible in some situations that negative FF gains

make sense, but in general all gains should be positive. If some output is in the wrong direction, negating gains to fix it is a mistake; set the scaling correctly elsewhere instead.

NAME

pluto_servo − Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with servo machines.

SYNOPSIS

loadrt pluto_servo [ioaddr=*N*] [ioaddr_hi=*N*] [epp_wide=*N*] [watchdog=*N*] [test_encoder=*N*]

ioaddr [default: 0x378]
: The base address of the parallel port.

ioaddr_hi [default: 0]
: The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400. -1 means there is no secondary address. The secondary address is used to set the port to EPP mode.

epp_wide [default: 1]
: Set to zero to disable the "wide EPP mode". "Wide" mode allows a 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this may not work on all EPP parallel ports.

watchdog [default: 1]
: Set to zero to disable the "hardware watchdog". "Watchdog" will tristate all outputs approximately 6ms after the last execution of **pluto-servo.write**, which adds some protection in the case of LinuxCNC crashes.

test_encoder [default: 0]
: Internally connect dout0..2 to QA0, QB0, QZ0 to test quadrature counting

DESCRIPTION

Pluto_servo is a LinuxCNC software driver and associated firmware that allow the Pluto-P board to be used to control a servo-based CNC machine.

The driver has 4 PWM channels, 4 quadrature channels with index pulse, 18 digital outputs (8 shared with PWM), and 20 digital inputs (12 shared with quadrature).

Encoders

The encoder pins and parameters conform to the ‘canonical encoder’ interface described in the HAL manual. It operates in ‘x4 mode’.

The sample rate of the encoder is 40MHz. The maximum number quadrature rate is 8191 counts per LinuxCNC servo cycle. For correct handling of the index pulse, the number of encoder counts per revolution must be less than 8191.

PWM

The PWM pins and parameters conform to the ‘canonical analog output’ interface described in the HAL manual. The output pins are ‘up/down’ or ‘pwm/dir’ pins as described in the documentation of the ‘pwmgen’ component.

Internally the PWM generator is based on a 12-bit, 40MHz counter, giving 4095 duty cycles from -100% to +100% and a frequency of approximately 19.5kHz. In PDM mode, the duty periods are approximately 100ns long.

Digital I/O

The digital output pins conform to the ‘canonical digital output’ interface described in the HAL manual.

The digital input pins conform to the ‘canonical digital input’ interface described in the HAL manual.

FUNCTIONS

pluto-servo.read (requires a floating-point thread)
Read all the inputs from the pluto-servo board

pluto-servo.write (requires a floating-point thread)
Write all the outputs on the pluto-servo board

PINS

pluto-servo.encoder.*M*.count s32 out (M=0..3)
pluto-servo.encoder.*M*.position float out (M=0..3)
pluto-servo.encoder.*M*.velocity float out (M=0..3)
pluto-servo.encoder.*M*.reset bit in (M=0..3)
pluto-servo.encoder.*M*.index-enable bit io (M=0..3)
encoder.*M* corresponds to the pins labeled QA*M*, QB*M*, and QZ*M* on the pinout diagram

pluto-servo.pwm.*M*.value float in (M=0..3)
pluto-servo.pwm.*M*.enable bit in (M=0..3)
pwm.*M* corresponds to the pins labeled UP*M* and DN*M* on the pinout diagram

pluto-servo.dout.*MM* bit in (MM=00..19)
dout.*0M* corresponds to the pin labeled OUT*M* on the pinout diagram. Other pins are shared with the PWM function, as follows:

Pin Label	**Shared with**
dout.10	UP0
dout.10	UP0
dout.12	UP1
dout.14	UP2
dout.18	UP3
dout.11	DOWN0
dout.13	DOWN1
dout.15	DOWN2
dout.19	DOWN3

pluto-servo.din.*MM* bit out (MM=00..19)
pluto-servo.din.*MM*-not bit out (MM=00..19)
For M=0 through 7, din.*0M* corresponds to the pin labeled IN*M* on the pinout diagram. Other pins are shared with the encoder function, as follows:

Pin Label	**Shared with**
din.8	QZ0
din.9	QZ1
din.10	QZ2
din.11	QZ3
din.12	QB0
din.13	QB1
din.14	QB2
din.15	QB3
din.16	QA0

din.17	QA1
din.18	QA2
din.19	QA3

PARAMETERS

pluto-servo.encoder.*M*.scale float rw (M=0..3) (default: *1*)

pluto-servo.encoder.z-polarity bit rw

Set to TRUE if the index pulse is active low, FALSE if it is active high. Affects all encoders.

pluto-servo.pwm.*M*.offset float rw (M=0..3)

pluto-servo.pwm.*M*.scale float rw (M=0..3) (default: *1*)

pluto-servo.pwm.*M*.max-dc float rw (M=0..3) (default: *1*)

pluto-servo.pwm.*M*.min-dc float rw (M=0..3) (default: *0*)

pluto-servo.pwm.*M*.pwmdir bit rw (M=0..3) (default: *0*)

Set to TRUE use PWM+direction mode. Set to FALSE to use Up/Down mode.

pluto-servo.pwm.is-pdm bit rw

Set to TRUE to use PDM (also called interleaved PWM) mode. Set to FALSE to use traditional PWM mode. Affects all PWM outputs.

pluto-servo.dout.*MM*-invert bit rw (MM=00..19)

If TRUE, the output on the corresponding **dout.*MM*** is inverted.

pluto-servo.communication-error u32 rw

Incremented each time pluto-servo.read detects an error code in the EPP status register. While this register is nonzero, new values are not being written to the Pluto-P board, and the status of digital outputs and the PWM duty cycle of the PWM outputs will remain unchanged. If the watchdog is enabled, it will activate soon after the communication error is detected. To continue after a communication error, set this parameter back to zero.

pluto-servo.debug-0 s32 rw

pluto-servo.debug-1 s32 rw

These parameters can display values which are useful to developers or for debugging the driver and firmware. They are not useful for integrators or users.

SEE ALSO

The *pluto_servo* section in the HAL User Manual, which shows the location of each physical pin on the pluto board.

LICENSE

GPL

NAME

pluto_step − Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with stepper machines.

SYNOPSIS

loadrt pluto_step ioaddr=*addr* **ioaddr_hi=***addr* **epp_wide=***[0|1]*

ioaddr [default: 0x378]
The base address of the parallel port.

ioaddr_hi [default: 0]
The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400. -1 means there is no secondary address.

epp_wide [default: 1]
Set to zero to disable "wide EPP mode". "Wide" mode allows 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this mode may not work on all EPP parallel ports.

watchdog [default: 1]
Set to zero to disable the "hardware watchdog". "Watchdog" will tristate all outputs approximately 6ms after the last execution of **pluto-step.write**, which adds some protection in the case of LinuxCNC crashes.

speedrange [default: 0]
Selects one of four speed ranges:
0: Top speed 312.5kHz; minimum speed 610Hz
1: Top speed 156.25kHz; minimum speed 305Hz
2: Top speed 78.125kHz; minimum speed 153Hz
3: Top speed 39.06kHz; minimum speed 76Hz
Choosing the smallest maximum speed that is above the maximum for any one axis may give improved step regularity at low step speeds.

DESCRIPTION

Pluto_step is a LinuxCNC software driver and associated firmware that allow the Pluto-P board to be used to control a stepper-based CNC machine.

The driver has 4 step+direction channels, 14 dedicated digital outputs, and 16 dedicated digital inputs.

Step generators

The step generator takes a position input and output.

The step waveform includes step length/space and direction hold/setup time. Step length and direction setup/hold time is enforced in the FPGA. Step space is enforced by a velocity cap in the driver.

(all the following numbers are subject to change) In *speedrange=0*, the maximum step rate is 312.5kHz. For position feedback to be accurate, the maximum step rate is 512 pulses per servo cycle (so a 1kHz servo cycle does not impose any additional limitation). The maximum step rate may be lowered by the step length and space parameters, which are rounded up to the nearest multiple of 1600ns.

In successive speedranges the maximum step rate is divided in half, as is the maximum steps per servo cycle, and the minimum nonzero step rate.

Digital I/O

The digital output pins conform to the ‘canonical digital output’ interface described in the HAL manual.

The digital input pins conform to the ‘canonical digital input’ interface described in the HAL manual.

FUNCTIONS

pluto-step.read (requires a floating-point thread)
Read all the inputs from the pluto-step board

pluto-step.write (requires a floating-point thread)
Write all the outputs on the pluto-step board

PINS

pluto-step.stepgen.*M*.position-cmd float in (M=0..3)
pluto-step.stepgen.*M*.velocity-fb float out (M=0..3)
pluto-step.stepgen.*M*.position-fb float out (M=0..3)
pluto-step.stepgen.*M*.counts s32 out (M=0..3)
pluto-step.stepgen.*M*.enable bit in (M=0..3)
pluto-step.stepgen.*M*.reset bit in (M=0..3)
When TRUE, reset position-fb to 0

pluto-step.dout.*MM* bit in (MM=00..13)
dout.*MM* corresponds to the pin labeled OUT*M* on the pinout diagram.

pluto-step.din.*MM* bit out (MM=00..15)
pluto-step.din.*MM*-not bit out (MM=00..15)
din.*MM* corresponds to the pin labeled IN*M* on the pinout diagram.

PARAMETERS

pluto-step.stepgen.*M*.scale float rw (M=0..3) (default: *1.0*)
pluto-step.stepgen.*M*.maxvel float rw (M=0..3) (default: *0*)
pluto-step.stepgen.step-polarity bit rw
pluto-step.stepgen.steplen u32 rw
Step length in ns.

pluto-step.stepgen.stepspace u32 rw
Step space in ns

pluto-step.stepgen.dirtime u32 rw
Dir hold/setup in ns. Refer to the pdf documentation for a diagram of what these timings mean.

pluto-step.dout.*MM*-invert bit rw (MM=00..13)
If TRUE, the output on the corresponding **dout.*MM*** is inverted.

pluto-step.communication-error u32 rw
Incremented each time pluto-step.read detects an error code in the EPP status register. While this register is nonzero, new values are not being written to the Pluto-P board, and the status of digital outputs and the PWM duty cycle of the PWM outputs will remain unchanged. If the hardware watchdog is enabled, it will activate shortly after the communication error is detected by LinuxCNC. To continue after a communication error, set this parameter back to zero.

pluto-step.debug-0 s32 rw
pluto-step.debug-1 s32 rw
pluto-step.debug-2 float rw (default: *.5*)
pluto-step.debug-3 float rw (default: *2.0*)
Registers that hold debugging information of interest to developers

SEE ALSO

The *pluto_step* section in the HAL User Manual, which shows the location of each physical pin on the pluto board.

LICENSE

GPL

NAME

pwmgen − software PWM/PDM generation

SYNOPSIS

loadrt pwmgen output_type=*type0*[,*type1*...]

DESCRIPTION

pwmgen is used to generate PWM (pulse width modulation) or PDM (pulse density modulation) signals. The maximum PWM frequency and the resolution is quite limited compared to hardware-based approaches, but in many cases software PWM can be very useful. If better performance is needed, a hardware PWM generator is a better choice.

pwmgen supports a maximum of eight channels. The number of channels actually loaded depends on the number of *type* values given. The value of each *type* determines the outputs for that channel.

type 0: single output

A single output pin, **pwm**, whose duty cycle is determined by the input value for positive inputs, and which is off (or at **min-dc**) for negative inputs. Suitable for single ended circuits.

type 1: pwm/direction

Two output pins, **pwm** and **dir**. The duty cycle on **pwm** varies as a function of the input value. **dir** is low for positive inputs and high for negative inputs.

type 2: up/down

Two output pins, **up** and **down**. For positive inputs, the PWM/PDM waveform appears on **up**, while **down** is low. For negative inputs, the waveform appears on **down**, while **up** is low. Suitable for driving the two sides of an H-bridge to generate a bipolar output.

FUNCTIONS

pwmgen.make-pulses (no floating-point)

Generates the actual PWM waveforms, using information computed by **update**. Must be called as frequently as possible, to maximize the attainable PWM frequency and resolution, and minimize jitter. Operates on all channels at once.

pwmgen.update (uses floating point)

Accepts an input value, performs scaling and limit checks, and converts it into a form usable by **make-pulses** for PWM/PDM generation. Can (and should) be called less frequently than **make-pulses**. Operates on all channels at once.

PINS

pwmgen.*N***.enable** bit in

Enables PWM generator *N* - when false, all **pwmgen.***N* output pins are low.

pwmgen.*N***.value** float in

Commanded value. When **value** = 0.0, duty cycle is 0%, and when **value** = +/-**scale**, duty cycle is +/- 100%. (Subject to **min-dc** and **max-dc** limitations.)

pwmgen.*N***.pwm** bit out (output types 0 and 1 only)

PWM/PDM waveform.

pwmgen.*N***.dir** bit out (output type 1 only)

Direction output: low for forward, high for reverse.

pwmgen.*N***.up** bit out (output type 2 only)

PWM/PDM waveform for positive input values, low for negative inputs.

pwmgen.*N***.down** bit out (output type 2 only)

PWM/PDM waveform for negative input values, low for positive inputs.

PARAMETERS

pwmgen.*N*.curr-dc float ro

The current duty cycle, after all scaling and limits have been applied. Range is from -1.0 to +1.0.

pwmgen.*N*.max-dc float rw

The maximum duty cycle. A value of 1.0 corresponds to 100%. This can be useful when using transistor drivers with bootstrapped power supplies, since the supply requires some low time to recharge.

pwmgen.*N*.min-dc float rw

The minimum duty cycle. A value of 1.0 corresponds to 100%. Note that when the pwm generator is disabled, the outputs are constantly low, regardless of the setting of **min-dc**.

pwmgen.*N*.scale float rw

pwmgen.*N*.offset float rw

These parameters provide a scale and offset from the **value** pin to the actual duty cycle. The duty cycle is calculated according to *dc = (value/scale) + offset*, with 1.0 meaning 100%.

pwmgen.*N*.pwm-freq float rw

PWM frequency in Hz. The upper limit is half of the frequency at which **make-pulses** is invoked, and values above that limit will be changed to the limit. If **dither-pwm** is false, the value will be changed to the nearest integer submultiple of the **make-pulses** frequency. A value of zero produces Pulse Density Modulation instead of Pulse Width Modulation.

pwmgen.*N*.dither-pwm bit rw

Because software-generated PWM uses a fairly slow timebase (several to many microseconds), it has limited resolution. For example, if **make-pulses** is called at a 20KHz rate, and **pwm-freq** is 2KHz, there are only 10 possible duty cycles. If **dither-pwm** is false, the commanded duty cycle will be rounded to the nearest of those values. Assuming **value** remains constant, the same output will repeat every PWM cycle. If **dither-pwm** is true, the output duty cycle will be dithered between the two closest values, so that the long-term average is closer to the desired level. **dither-pwm** has no effect if **pwm-freq** is zero (PDM mode), since PDM is an inherently dithered process.

NAME

sample_hold − Sample and Hold

SYNOPSIS

loadrt sample_hold [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

sample-hold.*N*

PINS

sample-hold.*N***.in** s32 in
sample-hold.*N***.hold** bit in
sample-hold.*N***.out** s32 out

LICENSE

GPL

NAME

sampler − sample data from HAL in real time

SYNOPSIS

loadrt sampler depth=*depth1[,depth2...]* **cfg=***string1[,string2...]*

DESCRIPTION

sampler and **halsampler**(1) are used together to sample HAL data in real time and store it in a file. **sampler** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. It then begins sampling data from the HAL and storing it to the FIFO. **halsampler** is a user space program that copies data from the FIFO to stdout, where it can be redirected to a file or piped to some other program.

OPTIONS

depth=*depth1[,depth2...]*

sets the depth of the realtime->user FIFO that **sampler** creates to buffer the realtime data. Multiple values of *depth* (separated by commas) can be specified if you need more than one FIFO (for example if you want to sample data from two different realtime threads).

cfg=*string1[,string2...]*

defines the set of HAL pins that **sampler** exports and later samples data from. One *string* must be supplied for each FIFO, separated by commas. **sampler** exports one pin for each character in *string.* Legal characters are:

F, f (float pin)

B, b (bit pin)

S, s (s32 pin)

U, u (u32 pin)

FUNCTIONS

sampler.*N*

One function is created per FIFO, numbered from zero.

PINS

sampler.*N***.pin.***M* input

Pin for the data that will wind up in column *M* of FIFO *N* (and in column *M* of the output file). The pin type depends on the config string.

sampler.*N***.curr-depth** s32 output

Current number of samples in the FIFO. When this reaches *depth* new data will begin overwriting old data, and some samples will be lost.

sampler.*N***.full** bit output

TRUE when the FIFO *N* is full, FALSE when there is room for another sample.

sampler.*N***.enable** bit input

When TRUE, samples are captured and placed in FIFO *N*, when FALSE, no samples are acquired. Defaults to TRUE.

PARAMETERS

sampler.*N***.overruns** s32 read/write

The number of times that **sampler** has tried to write data to the HAL pins but found no room in the FIFO. It increments whenever **full** is true, and can be reset by the **setp** command.

sampler.*N*.sample-num s32 read/write

A number that identifies the sample. It is automatically incremented for each sample, and can be reset using the **setp** command. The sample number can optionally be printed in the first column of the output from **halsampler**, using the *-t* option. (see **man 1 halsampler**)

SEE ALSO

halsampler(1) **streamer**(9) **halstreamer**(1)

HISTORY

BUGS

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

REPORTING BUGS

Report bugs to jmkasunich AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 John Kasunich.
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

scale − LinuxCNC HAL component that applies a scale and offset to its input

SYNOPSIS

loadrt scale [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

scale.*N* (requires a floating-point thread)

PINS

scale.*N***.in** float in
scale.*N***.gain** float in
scale.*N***.offset** float in
scale.*N***.out** float out
out = in * gain + offset

LICENSE

GPL

NAME

select8 − 8-bit binary match detector

SYNOPSIS

loadrt select8 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

select8.*N*

PINS

select8.*N***.sel** s32 in
: The number of the output to set TRUE. All other outputs well be set FALSE

select8.*N***.out***M* bit out (M=0..7)
: Output bits. If enable is set and the sel input is between 0 and 7, then the corresponding output bit will be set true

PARAMETERS

select8.*N***.enable** bit rw (default: *TRUE*)
: Set enable to FALSE to cause all outputs to be set FALSE

LICENSE

GPL

NAME

serport − Hardware driver for the digital I/O bits of the 8250 and 16550 serial port.

SYNOPSIS

loadrt serport io=*addr[,addr...]*

The pin numbers refer to the 9-pin serial pinout. Keep in mind that these ports generally use rs232 voltages, not 0/5V signals.

Specify the I/O address of the serial ports using the module parameter **io=***addr[,addr...]*. These ports must not be in use by the kernel. To free up the I/O ports after bootup, install setserial and execute a command like:

sudo setserial /dev/ttyS0 uart none

but it is best to ensure that the serial port is never used or configured by the Linux kernel by setting a kernel commandline parameter or not loading the serial kernel module if it is a modularized driver.

FUNCTIONS

serport.*N***.read**

serport.*N***.write**

PINS

serport.*N***.pin-1-in** bit out
Also called DCD (data carrier detect); pin 8 on the 25-pin serial pinout

serport.*N***.pin-6-in** bit out
Also called DSR (data set ready); pin 6 on the 25-pin serial pinout

serport.*N***.pin-8-in** bit out
Also called CTS (clear to send); pin 5 on the 25-pin serial pinout

serport.*N***.pin-9-in** bit out
Also called RI (ring indicator); pin 22 on the 25-pin serial pinout

serport.*N***.pin-1-in-not** bit out
Inverted version of pin-1-in

serport.*N***.pin-6-in-not** bit out
Inverted version of pin-6-in

serport.*N***.pin-8-in-not** bit out
Inverted version of pin-8-in

serport.*N***.pin-9-in-not** bit out
Inverted version of pin-9-in

serport.*N***.pin-3-out** bit in
Also called TX (transmit data); pin 2 on the 25-pin serial pinout

serport.*N***.pin-4-out** bit in
Also called DTR (data terminal ready); pin 20 on the 25-pin serial pinout

serport.*N***.pin-7-out** bit in
Also called RTS (request to send); pin 4 on the 25-pin serial pinout

PARAMETERS

serport.*N***.pin-3-out-invert** bit rw
serport.*N***.pin-4-out-invert** bit rw
serport.*N***.pin-7-out-invert** bit rw

serport.*N*.ioaddr u32 r

LICENSE

GPL

NAME

siggen − signal generator

SYNOPSIS

loadrt siggen [num_chan=*num* **| names=***name1***[,***name2...***]]**

DESCRIPTION

siggen is a signal generator that can be used for testing and other applications that need simple waveforms. It produces sine, cosine, triangle, sawtooth, and square waves of variable frequency, amplitude, and offset, which can be used as inputs to other HAL components.

siggen supports a maximum of sixteen channels. The number of channels actually loaded is set by the **num_chan** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is one.

NAMING

The names for pins, parameters, and functions are prefixed as:
siggen.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **siggen.N.** format is shown in the following descriptions.

FUNCTIONS

siggen.*N*.update (uses floating-point)

Updates output pins for signal generator *N*. Each time it is called it calculates a new sample. It should be called many times faster than the desired signal frequency, to avoid distortion and aliasing.

PINS

siggen.*N*.frequency float in

The output frequency for signal generator *N*, in Hertz. The default value is 1.0 Hertz.

siggen.*N*.amplitude float in

The output amplitude for signal generator *N*. If **offset** is zero, the outputs will swing from -**amplitude** to +**amplitude**. The default value is 1.00.

siggen.*N*.offset float in

The output offset for signal generator *N*. This value is added directly to the output signal. The default value is zero.

siggen.*N*.clock bit out

The clock output. Bit type clock signal output at the commanded frequency.

siggen.*N*.square float out

The square wave output. Positive while **triangle** and **cosine** are ramping upwards, and while **sine** is negative.

siggen.*N*.sine float out

The sine output. Lags **cosine** by 90 degrees.

siggen.*N*.cosine float out
The cosine output. Leads **sine** by 90 degrees.

siggen.*N*.triangle float out
The triangle wave output. Ramps up while **square** is positive, and down while **square** is negative. Reaches its positive and negative peaks at the same time as **cosine**.

siggen.*N*.sawtooth float out
The sawtooth output. Ramps upwards to its positive peak, then instantly drops to its negative peak and starts ramping again. The drop occurs when **triangle** and **cosine** are at their positive peaks, and coincides with the falling edge of **square**.

PARAMETERS

None

NAME

sim_encoder − simulated quadrature encoder

SYNOPSIS

loadrt sim_encoder [num_chan=*num* **| names=***name1***[,***name2***...]]**

DESCRIPTION

sim_encoder can generate quadrature signals as if from an encoder. It also generates an index pulse once per revolution. It is mostly used for testing and simulation, to replace hardware that may not be available. It has a limited maximum frequency, as do all software based pulse generators.

sim_encoder supports a maximum of eight channels. The number of channels actually loaded is set by the **num_chan=** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num_chan=** and **names=** specifiers are mutually exclusive. If neither **num_chan=** nor **names=** are specified, the default value is one.

FUNCTIONS

sim-encoder.make-pulses (no floating-point)

Generates the actual quadrature and index pulses. Must be called as frequently as possible, to maximize the count rate and minimize jitter. Operates on all channels at once.

sim-encoder.update-speed (uses floating-point)

Reads the **speed** command and other parameters and converts the data into a form that can be used by **make-pulses**. Changes take effect only when **update-speed** runs. Can (and should) be called less frequently than **make-pulses**. Operates on all channels at once.

NAMING

The names for pins and parameters are prefixed as:
sim-encoder.N. for N=0,1,...,num-1 when using **num_chan=num**
nameN. for nameN=name1,name2,... when using **names=name1,name2,...**

The **sim-encoder.N.** format is shown in the following descriptions.

PINS

sim-encoder.*N*.phase-A bit out

One of the quadrature outputs.

sim-encoder.*N*.phase-B bit out

The other quadrature output.

sim-encoder.*N*.phase-Z bit out

The index pulse.

sim-encoder.*N*.speed float in

The desired speed of the encoder, in user units per per second. This is divided by **scale**, and the result is used as the encoder speed in revolutions per second.

PARAMETERS

sim-encoder.*N*.ppr u32 rw

The pulses per revolution of the simulated encoder. Note that this is pulses, not counts, per revolution. Each pulse or cycle from the encoder results in four counts, because every edge is counted. Default value is 100 ppr, or 400 counts per revolution.

sim-encoder.*N*.scale float rw

Scale factor for the **speed** input. The **speed** value is divided by **scale** to get the actual encoder speed in revolutions per second. For example, if **scale** is set to 60, then **speed** is in revolutions per minute (RPM) instead of revolutions per second. The default value is 1.00.

NAME

sphereprobe − Probe a pretend hemisphere

SYNOPSIS

loadrt sphereprobe [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

sphereprobe.*N*

update probe-out based on inputs

PINS

sphereprobe.*N***.px** s32 in
sphereprobe.*N***.py** s32 in
sphereprobe.*N***.pz** s32 in

rawcounts position from software encoder

sphereprobe.*N***.cx** s32 in
sphereprobe.*N***.cy** s32 in
sphereprobe.*N***.cz** s32 in

Center of sphere in counts

sphereprobe.*N***.r** s32 in

Radius of hemisphere in counts

sphereprobe.*N***.probe-out** bit out

AUTHOR

Jeff Epler

LICENSE

GPL

NAME

stepgen − software step pulse generation

SYNOPSIS

loadrt stepgen step_type=*type0*[,*type1*...] [**ctrl_type=***type0*[,*type1*...]] [**user_step_type=#,#**...]

DESCRIPTION

stepgen is used to control stepper motors. The maximum step rate depends on the CPU and other factors, and is usually in the range of 5KHz to 25KHz. If higher rates are needed, a hardware step generator is a better choice.

stepgen has two control modes, which can be selected on a channel by channel basis using **ctrl_type**. Possible values are "**p**" for position control, and "**v**" for velocity control. The default is position control, which drives the motor to a commanded position, subject to acceleration and velocity limits. Velocity control drives the motor at a commanded speed, again subject to accel and velocity limits. Usually, position mode is used for machine axes. Velocity mode is reserved for unusual applications where continuous movement at some speed is desired, instead of movement to a specific position. (Note that velocity mode replaces the former component **freqgen**.)

stepgen can control a maximum of eight motors. The number of motors/channels actually loaded depends on the number of *type* values given. The value of each *type* determines the outputs for that channel. Position or velocity mode can be individually selected for each channel. Both control modes support the same 16 possible step types.

By far the most common step type is ’0’, standard step and direction. Others include up/down, quadrature, and a wide variety of three, four, and five phase patterns that can be used to directly control some types of motor windings. (When used with appropriate buffers of course.)

Some of the stepping types are described below, but for more details (including timing diagrams) see the **stepgen** section of the HAL reference manual.

type 0: step/dir
: Two pins, one for step and one for direction. **make-pulses** must run at least twice for each step (once to set the step pin true, once to clear it). This limits the maximum step rate to half (or less) of the rate that can be reached by types 2-14. The parameters **steplen** and **stepspace** can further lower the maximum step rate. Parameters **dirsetup** and **dirhold** also apply to this step type.

type 1: up/down
: Two pins, one for ’step up’ and one for ’step down’. Like type 0, **make-pulses** must run twice per step, which limits the maximum speed.

type 2: quadrature
: Two pins, phase-A and phase-B. For forward motion, A leads B. Can advance by one step every time **make-pulses** runs.

type 3: three phase, full step
: Three pins, phase-A, phase-B, and phase-C. Three steps per full cycle, then repeats. Only one phase is high at a time - for forward motion the pattern is A, then B, then C, then A again.

type 4: three phase, half step
: Three pins, phases A through C. Six steps per full cycle. First A is high alone, then A and B together, then B alone, then B and C together, etc.

types 5 through 8: four phase, full step
: Four pins, phases A through D. Four steps per full cycle. Types 5 and 6 are suitable for use with unipolar steppers, where power is applied to the center tap of each winding, and four open-collector transistors drive the ends. Types 7 and 8 are suitable for bipolar steppers, driven by two H-bridges.

types 9 and 10: four phase, half step
: Four pins, phases A through D. Eight steps per full cycle. Type 9 is suitable for unipolar drive, and type 10 for bipolar drive.

types 11 and 12: five phase, full step
: Five pins, phases A through E. Five steps per full cycle. See HAL reference manual for the patterns.

types 13 and 14: five phase, half step
: Five pins, phases A through E. Ten steps per full cycle. See HAL reference manual for the patterns.

type 15: user-specified
: This uses the waveform specified by the **user_step_type** module parameter, which may have up to 10 steps and 5 phases.

FUNCTIONS

stepgen.make-pulses (no floating-point)
: Generates the step pulses, using information computed by **update-freq**. Must be called as frequently as possible, to maximize the attainable step rate and minimize jitter. Operates on all channels at once.

stepgen.capture-position (uses floating point)
: Captures position feedback value from the high speed code and makes it available on a pin for use elsewhere in the system. Operates on all channels at once.

stepgen.update-freq (uses floating point)
: Accepts a velocity or position command and converts it into a form usable by **make-pulses** for step generation. Operates on all channels at once.

PINS

stepgen.*N*.counts s32 out
: The current position, in counts, for channel *N*. Updated by **capture-position**.

stepgen.*N*.position-fb float out
: The current position, in length units (see parameter **position-scale**). Updated by **capture-position**. The resolution of **position-fb** is much finer than a single step. If you need to see individual steps, use **counts**.

stepgen.*N*.enable bit in
: Enables output steps - when false, no steps are generated.

stepgen.*N*.velocity-cmd float in (velocity mode only)
: Commanded velocity, in length units per second (see parameter **position-scale**).

stepgen.*N*.position-cmd float in (position mode only)
: Commanded position, in length units (see parameter **position-scale).**

stepgen.*N*.step bit out (step type 0 only)
: Step pulse output.

stepgen.*N*.dir bit out (step type 0 only)
: Direction output: low for forward, high for reverse.

stepgen.*N*.up bit out (step type 1 only)
: Count up output, pulses for forward steps.

stepgen.*N*.down bit out (step type 1 only)
: Count down output, pulses for reverse steps.

stepgen.*N*.phase-A thru **phase-E** bit out (step types 2-14 only)
: Output bits. **phase-A** and **phase-B** are present for step types 2-14, **phase-C** for types 3-14, **phase-D** for types 5-14, and **phase-E** for types 11-14. Behavior depends on selected stepping type.

PARAMETERS

stepgen.*N*.frequency float ro
The current step rate, in steps per second, for channel *N*.

stepgen.*N*.maxaccel float rw
The acceleration/deceleration limit, in length units per second squared.

stepgen.*N*.maxvel float rw
The maximum allowable velocity, in length units per second. If the requested maximum velocity cannot be reached with the current combination of scaling and **make-pulses** thread period, it will be reset to the highest attainable value.

stepgen.*N*.position-scale float rw
The scaling for position feedback, position command, and velocity command, in steps per length unit.

stepgen.*N*.rawcounts s32 ro
The position in counts, as updated by **make-pulses**. (Note: this is updated more frequently than the **counts** pin.)

stepgen.*N*.steplen u32 rw
The length of the step pulses, in nanoseconds. Measured from rising edge to falling edge.

stepgen.*N*.stepspace u32 rw (step types 0 and 1 only) The minimum
space between step pulses, in nanoseconds. Measured from falling edge to rising edge. The actual time depends on the step rate and can be much longer. If **stepspace** is 0, then **step** can be asserted every period. This can be used in conjunction with **hal_parport**'s auto-resetting pins to output one step pulse per period. In this mode, **steplen** must be set for one period or less.

stepgen.*N*.dirsetup u32 rw (step type 0 only)
The minimum setup time from direction to step, in nanoseconds periods. Measured from change of direction to rising edge of step.

stepgen.*N*.dirhold u32 rw (step type 0 only)
The minimum hold time of direction after step, in nanoseconds. Measured from falling edge of step to change of direction.

stepgen.*N*.dirdelay u32 rw (step types 1 and higher only)
The minimum time between a forward step and a reverse step, in nanoseconds.

TIMING

There are five timing parameters which control the output waveform. No step type uses all five, and only those which will be used are exported to HAL. The values of these parameters are in nano-seconds, so no recalculation is needed when changing thread periods. In the timing diagrams that follow, they are identified by the following numbers:

(1) **stepgen.n.steplen**

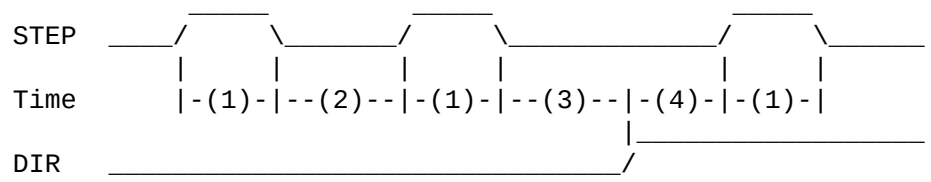
(2) **stepgen.n.stepspace**

(3) **stepgen.n.dirhold**

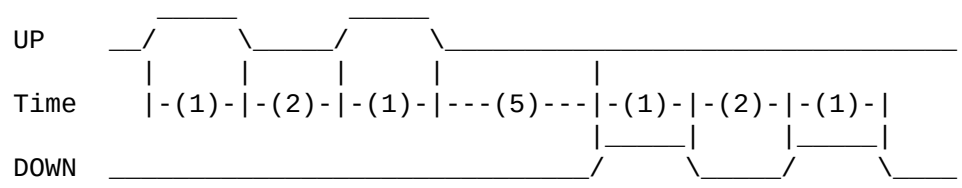
(4) **stepgen.n.dirsetup**

(5) **stepgen.n.dirdelay**

For step type 0, timing parameters 1 thru 4 are used. The following timing diagram shows the output waveforms, and what each parameter adjusts.



For step type 1, timing parameters 1, 2, and 5 are used. The following timing diagram shows the output waveforms, and what each parameter adjusts.



For step types 2 and higher, the exact pattern of the outputs depends on the step type (see the HAL manual for a full listing). The outputs change from one state to another at a minimum interval of **steplen**. When a direction change occurs, the minimum time between the last step in one direction and the first in the other direction is the sum of **steplen** and **dirdelay**.

SEE ALSO

The HAL User Manual.

NAME

steptest − Used by Stepconf to allow testing of acceleration and velocity values for an axis.

SYNOPSIS

loadrt steptest [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

steptest.*N* (requires a floating-point thread)

PINS

steptest.*N***.jog-minus** bit in
Drive TRUE to jog the axis in its minus direction

steptest.*N***.jog-plus** bit in
Drive TRUE to jog the axis in its positive direction

steptest.*N***.run** bit in
Drive TRUE to run the axis near its current position_fb with a trapezoidal velocity profile

steptest.*N***.maxvel** float in
Maximum velocity

steptest.*N***.maxaccel** float in
Permitted Acceleration

steptest.*N***.amplitude** float in
Approximate amplitude of positions to command during ’run’

steptest.*N***.dir** s32 in
Direction from central point to test: 0 = both, 1 = positive, 2 = negative

steptest.*N***.position-cmd** float out
steptest.*N***.position-fb** float in
steptest.*N***.running** bit out
steptest.*N***.run-target** float out
steptest.*N***.run-start** float out
steptest.*N***.run-low** float out
steptest.*N***.run-high** float out
steptest.*N***.pause** s32 in (default: *0*)
pause time for each end of run in seconds

PARAMETERS

steptest.*N***.epsilon** float rw (default: *.001*)
steptest.*N***.elapsed** float r
Current value of the internal timer

LICENSE

GPL

NAME

streamer − stream file data into HAL in real time

SYNOPSIS

loadrt streamer depth=*depth1[,depth2...]* **cfg=***string1[,string2...]*

DESCRIPTION

streamer and **halstreamer**(1) are used together to stream data from a file into the HAL in real time. **streamer** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. **hal_streamer** is a user space program that copies data from stdin into the FIFO, so that **streamer** can write it to the HAL pins.

OPTIONS

depth=*depth1[,depth2...]*

sets the depth of the user->realtime FIFO that **streamer** creates to receive data from **halstreamer.** Multiple values of *depth* (separated by commas) can be specified if you need more than one FIFO (for example if you want to stream data from two different realtime threads).

cfg=*string1[,string2...]*

defines the set of HAL pins that **streamer** exports and later writes data to. One *string* must be supplied for each FIFO, separated by commas. **streamer** exports one pin for each character in *string.* Legal characters are:

F, f (float pin)

B, b (bit pin)

S, s (s32 pin)

U, u (u32 pin)

FUNCTIONS

streamer.*N*

One function is created per FIFO, numbered from zero.

PINS

streamer.*N***.pin.***M* output

Data from column *M* of the data in FIFO *N* appears on this pin. The pin type depends on the config string.

streamer.*N***.curr-depth** s32 output

Current number of samples in the FIFO. When this reaches zero, new data will no longer be written to the pins.

streamer.*N***.empty** bit output

TRUE when the FIFO *N* is empty, FALSE when valid data is available.

streamer.*N***.enable** bit input

When TRUE, data from FIFO *N* is written to the HAL pins. When false, no data is transferred. Defaults to TRUE.

streamer.*N***.underruns** s32 read/write

The number of times that **sampler** has tried to write data to the HAL pins but found no fresh data in the FIFO. It increments whenever **empty** is true, and can be reset by the **setp** command.

SEE ALSO

halstreamer(1) **sampler**(9) **halsampler**(1)

HISTORY

BUGS

Should an **enable** HAL pin be added, to allow streaming to be turned on and off?

AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

REPORTING BUGS

Report bugs to jmkasunich AT users DOT sourceforge DOT net

COPYRIGHT

Copyright © 2006 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

NAME

sum2 − Sum of two inputs (each with a gain) and an offset

SYNOPSIS

loadrt sum2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

sum2.*N* (requires a floating-point thread)

PINS

sum2.*N***.in0** float in
sum2.*N***.in1** float in
sum2.*N***.out** float out
out = in0 * gain0 + in1 * gain1 + offset

PARAMETERS

sum2.*N***.gain0** float rw (default: *1.0*)
sum2.*N***.gain1** float rw (default: *1.0*)
sum2.*N***.offset** float rw

LICENSE

GPL

NAME

supply − set output pins with values from parameters (obsolete)

SYNOPSIS

loadrt supply num_chan=*num*

DESCRIPTION

supply was used to allow the inputs of other HAL components to be manipulated for testing purposes. When it was written, the only way to set the value of an input pin was to connect it to a signal and connect that signal to an output pin of some other component, and then let that component write the pin value. **supply** was written to be that "other component". It reads values from parameters (set with the HAL command **setp**) and writes them to output pins.

Since **supply** was written, the **setp** command has been modified to allow it to set unconnected pins as well as parameters. In addition, the **sets** command was added, which can directly set HAL signals, as long as there are no output pins connected to them. Therefore, **supply** is obsolete.

supply supports a maximum of eight channels. The number of channels actually loaded is set by the **num_chan** argument when the module is loaded. If **numchan** is not specified, the default value is one.

FUNCTIONS

supply.*N***.update** (uses floating-point)
Updates output pins for channel *N*.

PINS

supply.*N***.q** bit out
Output bit, copied from parameter **supply.***N***.d**.

supply.*N***._q** bit out
Output bit, inverted copy of parameter **supply.***N***.d**.

supply.*N***.variable** float out
Analog output, copied from parameter **supply.***N***.value**.

supply.*N***._variable** float out
Analog output, equal to -1.0 times parameter **supply.***N***.value**.

supply.*N***.d** bit rw
Data source for **q** and **_q** output pins.

supply.*N***.value** bit rw
Data source for **variable** and **_variable** output pins.

NAME

thc − Torch Height Control

SYNOPSIS

loadrt thc

DESCRIPTION

Torch Height Control Mesa THC > Encoder > LinuxCNC THC component

The Mesa THC sends a frequency based on the voltage detected to the encoder. The velocity from the encoder is converted to volts with the velocity scale parameter inside the THC component.

The THCAD card sends a frequency at 0 volts so the scale offset parameter is used to zero the calculated voltage.

Component Functions If enabled and torch is on and X + Y velocity is within tolerance of set speed allow the THC to offset the Z axis as needed to maintain voltage.

If enabled and torch is off and the Z axis is moving up remove any correction at a rate not to exceed the rate of movement of the Z axis.

If enabled and torch is off and there is no correction pass the Z position and feed back untouched.

If not enabled pass the Z position and feed back untouched.

Physical Connections
Plasma Torch Arc Voltage Signal => 6 x 487k 1% resistors => THC Arc Voltage In
THC Frequency Signal => Encoder #0, pin A (Input)
Plasma Torch Arc OK Signal => input pin
output pin => Plasma Torch Start Arc Contacts

HAL Plasma Connections
encoder.nn.velocity => thc.encoder-vel (tip voltage)
motion.spindle-on => output pin (start the arc)
thc.arc-ok <= motion.digital-in-00 <= input pin (arc ok signal)

HAL Motion Connections
thc.requested-vel <= motion.requested-vel
thc.current-vel <= motion.current-vel

FUNCTIONS

thc (requires a floating-point thread)

PINS

thc.encoder-vel float in
Connect to hm2_5i20.0.encoder.00.velocity

thc.current-vel float in
Connect to motion.current-vel

thc.requested-vel float in
Connect to motion.requested-vel

thc.volts-requested float in
Tip Volts current_vel >= min_velocity requested

thc.vel-tol float in
Velocity Tolerance (Corner Lock)

thc.torch-on bit in
Connect to motion.spindle-on

thc.arc-ok bit in
Arc OK from Plasma Torch

thc.enable bit in
Enable the THC, if not enabled Z position is passed through

thc.z-pos-in float in
Z Motor Position Command in from axis.n.motor-pos-cmd

thc.z-pos-out float out
Z Motor Position Command Out

thc.z-fb-out float out
Z Position Feedback to Axis

thc.volts float out
The Calculated Volts

thc.vel-status bit out
When the THC thinks we are at requested speed

PARAMETERS

thc.vel-scale float rw
The scale to convert the Velocity signal to Volts

thc.scale-offset float rw
The offset of the velocity input at 0 volts

thc.velocity-tol float rw
The deviation percent from planned velocity

thc.voltage-tol float rw
The deviation of Tip Voltage before correction takes place

thc.correction-vel float rw
The amount of change in user units per period to move Z to correct

AUTHOR

John Thornton

LICENSE

GPLv2 or greater

NAME

threads − creates hard realtime HAL threads

SYNOPSIS

loadrt threads name1=*name* **period1=***period* [**fp1=**<**0**|**1**>] [<thread-2-info>] [<thread-3-info>]

DESCRIPTION

threads is used to create hard realtime threads which can execute HAL functions at specific intervals. It is not a true HAL component, in that it does not export any functions, pins, or parameters of its own. Once it has created one or more threads, the threads stand alone, and the **threads** component can be unloaded without affecting them. In fact, it can be unloaded and then reloaded to create additional threads, as many times as needed.

threads can create up to three realtime threads. Threads must be created in order, from fastest to slowest. Each thread is specified by three arguments. **name1** is used to specify the name of the first thread (thread 1). **period1** is used to specify the period of thread 1 in nanoseconds. Both *name* and *period* are required. The third argument, **fp1** is optional, and is used to specify if thread 1 will be used to execute floating point code. If not specified, it defaults to **1**, which means that the thread will support floating point. Specify **0** to disable floating point support, which saves a small amount of execution time by not saving the FPU context. For additional threads, **name2**, **period2**, **fp2**, **name3**, **period3**, and **fp3** work exactly the same. If more than three threads are needed, unload threads, then reload it to create more threads.

FUNCTIONS

None

PINS

None

PARAMETERS

None

BUGS

The existence of **threads** might be considered a bug. Ideally, creation and deletion of threads would be done directly with **halcmd** commands, such as "**newthread** *name period*", "**delthread** *name*", or similar. However, limitations in the current HAL implementation require thread creation to take place in kernel space, and loading a component is the most straightforward way to do that.

NAME

threadtest − LinuxCNC HAL component for testing thread behavior

SYNOPSIS

loadrt threadtest [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

threadtest.*N*.increment

threadtest.*N*.reset

PINS

threadtest.*N*.count u32 out

LICENSE

GPL

NAME

time − Time on in Hours, Minutes, Seconds

SYNOPSIS

loadrt time [count=*N***|names=***name1***[,***name2***...]]**

DESCRIPTION

Time

When the time.N.start bit goes true the cycle timer resets and starts to time until time.N.start goes false. If you connect time.N.start to halui.is-running as a cycle timer it will reset during a pause. See the example connections below to keep the timer timing during a pause.

Time returns the hours, minutes, and seconds that time.N.start is true.

Sample pyVCP code to display the hours:minutes:seconds.

```
<pyvcp>
 <hbox>
 <label>
   <text>"Cycle Time"</text>
   <font>("Helvetica",14)</font>
 </label>
 <u32>
    <halpin>"time-hours"</halpin>
    <font>("Helvetica",14)</font>
    <format>"2d"</format>
 </u32>
 <label>
   <text>":"</text>
   <font>("Helvetica",14)</font>
 </label>
 <u32>
    <halpin>"time-minutes"</halpin>
    <font>("Helvetica",14)</font>
    <format>"2d"</format>
 </u32>
 <label>
   <text>":"</text>
   <font>("Helvetica",14)</font>
 </label>
 <u32>
    <halpin>"time-seconds"</halpin>
    <font>("Helvetica",14)</font>
    <format>"2d"</format>
 </u32>
 </hbox> </pyvcp>
```

In your post-gui.hal file you might use the following to connect it up

```
loadrt time
loadrt not
addf time.0 servo-thread
addf not.0 servo-thread
net prog-running not.0.in <= halui.program.is-idle
net cycle-timer time.0.start <= not.0.out
```

```
net cycle-seconds pyvcp.time-seconds <= time.0.seconds
net cycle-minutes pyvcp.time-minutes <= time.0.minutes
net cycle-hours pyvcp.time-hours <= time.0.hours
```

FUNCTIONS

time.*N* (requires a floating-point thread)

PINS

time.*N***.start** bit in
Timer On

time.*N***.seconds** u32 out
Seconds

time.*N***.minutes** u32 out
Minutes

time.*N***.hours** u32 out
Hours

AUTHOR

John Thornton

LICENSE

GPL

NAME

timedelay − The equivalent of a time-delay relay

SYNOPSIS

loadrt timedelay [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

timedelay.*N* (requires a floating-point thread)

PINS

timedelay.*N***.in** bit in

timedelay.*N***.out** bit out

Follows the value of **in** after applying the delays **on-delay** and **off-delay**.

timedelay.*N***.on-delay** float in (default: *0.5*)

The time, in seconds, for which **in** must be **true** before **out** becomes **true**

timedelay.*N***.off-delay** float in (default: *0.5*)

The time, in seconds, for which **in** must be **false** before **out** becomes **false**

timedelay.*N***.elapsed** float out

Current value of the internal timer

AUTHOR

Jeff Epler, based on works by Stephen Wille Padnos and John Kasunich

LICENSE

GPL

NAME

timedelta − LinuxCNC HAL component that measures thread scheduling timing behavior

SYNOPSIS

loadrt timedelta [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

timedelta.*N*

PINS

timedelta.*N***.out** s32 out
timedelta.*N***.err** s32 out (default: *0*)
timedelta.*N***.min** s32 out (default: *0*)
timedelta.*N***.max** s32 out (default: *0*)
timedelta.*N***.jitter** s32 out (default: *0*)
timedelta.*N***.avg-err** float out (default: *0*)
timedelta.*N***.reset** bit in

LICENSE

GPL

NAME

toggle − ’push-on, push-off’ from momentary pushbuttons

SYNOPSIS

loadrt toggle [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

toggle.*N*

PINS

toggle.*N***.in** bit in
button input

toggle.*N***.out** bit io
on/off output

PARAMETERS

toggle.*N***.debounce** u32 rw (default: *2*)
debounce delay in periods

LICENSE

GPL

NAME

toggle2nist − toggle button to nist logic

SYNOPSIS

loadrt toggle2nist [count=*N***|names=***name1***[,***name2...***]]**

DESCRIPTION

toggle2nist can be used with a momentary push button connected to a toggle component to control a device that has seperate on and off inputs and has an is-on output. If in changes states via the toggle output
If is-on is true then on is false and off is true.
If is-on is false the on true and off is false.

FUNCTIONS

toggle2nist.*N* (requires a floating-point thread)

PINS

toggle2nist.*N***.in** bit in
toggle2nist.*N***.is-on** bit in
toggle2nist.*N***.on** bit out
toggle2nist.*N***.off** bit out

LICENSE

GPL

NAME

tristate_bit − Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics

SYNOPSIS

loadrt tristate_bit [count=*N*|**names=***name1*[**,***name2...*]]

FUNCTIONS

tristate-bit.*N*

If **enable** is TRUE, copy **in** to **out**.

PINS

tristate-bit.*N***.in** bit in

Input value

tristate-bit.*N***.out** bit io

Output value

tristate-bit.*N***.enable** bit in

When TRUE, copy in to out

LICENSE

GPL

NAME

tristate_float − Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics

SYNOPSIS

loadrt tristate_float [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

tristate-float.*N* (requires a floating-point thread)
If **enable** is TRUE, copy **in** to **out**.

PINS

tristate-float.*N***.in** float in
Input value

tristate-float.*N***.out** float io
Output value

tristate-float.*N***.enable** bit in
When TRUE, copy in to out

LICENSE

GPL

NAME

updown − Counts up or down, with optional limits and wraparound behavior

SYNOPSIS

loadrt updown [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

updown.*N*

Process inputs and update count if necessary

PINS

updown.*N***.countup** bit in

Increment count when this pin goes from 0 to 1

updown.*N***.countdown** bit in

Decrement count when this pin goes from 0 to 1

updown.*N***.reset** bit in

Reset count when this pin goes from 0 to 1

updown.*N***.count** s32 out

The current count

PARAMETERS

updown.*N***.clamp** bit rw

If TRUE, then clamp the output to the min and max parameters.

updown.*N***.wrap** bit rw

If TRUE, then wrap around when the count goes above or below the min and max parameters. Note that wrap implies (and overrides) clamp.

updown.*N***.max** s32 rw (default: *0x7FFFFFFF*)

If clamp or wrap is set, count will never exceed this number

updown.*N***.min** s32 rw

If clamp or wrap is set, count will never be less than this number

LICENSE

GPL

NAME

watchdog − monitor multiple inputs for a "heartbeat"

SYNOPSIS

loadrt watchdog num_inputs=*N*

You must specify the number of inputs, from 1 to 32. Each input has a separate timeout value.

FUNCTIONS

process

Check all input pins for transitions, clear the **ok-out** pin if any input has no transition within its timeout period. This function does not use floating point, and should be added to a fast thread.

set-timeouts

Check for timeout changes, and convert the float timeout inputs to int values that can be used in **process**. This function also monitors **enable-in** for false to true transitions, and re-enables monitoring when such a transition is detected. This function does use floating point, and it is appropriate to add it to the servo thread.

PINS

watchdog.input-n bit in

Input number n. The inputs are numbered from 0 to **num_inputs**-1.

watchdog.enable-in bit in (default: *FALSE*)

If TRUE, forces out-ok to be false. Additionally, if a timeout occurs on any input, this pin must be set FALSE and TRUE again to re-start the monitoring of input pins.

watchdog.ok-out bit out (default: *FALSE*)

OK output. This pin is true only if enable-in is TRUE and no timeout has been detected. This output can be connected to the enable input of a **charge_pump** or **stepgen** (in v mode), to provide a heartbeat signal to external monitoring hardware.

PARAMETERS

watchdog.timeout-n float in

Timeout value for input number n. The inputs are numbered from 0 to **num_inputs**-1. The timeout is in seconds, and may not be below zero. Note that a timeout of 0.0 will likely prevent **ok-out** from ever becoming true. Also note that excessively long timeouts are relatively useless for monitoring purposes.

LICENSE

GPL

NAME

wcomp − Window comparator

SYNOPSIS

loadrt wcomp [count=*N***|names=***name1***[,***name2...***]]**

FUNCTIONS

wcomp.*N* (requires a floating-point thread)

PINS

wcomp.*N***.in** float in
: Value being compared

wcomp.*N***.min** float in
: Low boundary for comparison

wcomp.*N***.max** float in
: High boundary for comparison

wcomp.*N***.out** bit out
: True if **in** is strictly between **min** and **max**

wcomp.*N***.under** bit out
: True if **in** is less than or equal to **min**

wcomp.*N***.over** bit out
: True if **in** is greater than or equal to **max**

NOTES

If **max** <= **min** then the behavior is undefined.

LICENSE

GPL

NAME

weighted_sum − convert a group of bits to an integer

SYNOPSIS

loadrt weighted_sum wsum_sizes=*size[,size,...]*

Creates weighted sum groups each with the given number of input bits (*size*).

DESCRIPTION

This component is a "weighted summer": Its output is the offset plus the sum of the weight of each TRUE input bit. The default value for each weight is 2ˆn where n is the bit number. This results in a binary to unsigned conversion.

There is a limit of 8 weighted summers and each may have up to 16 input bits.

FUNCTIONS

process_wsums (requires a floating point thread)

Read all input values and update all output values.

PINS

wsum.*N***.bit.***M***.in** bit in

The *m*'th input of weighted summer *n*.

wsum.*N***.hold** bit in

When TRUE, the *sum* output does not change. When FALSE, the *sum* output tracks the *bit* inputs according to the weights and offset.

wsum.*N***.sum** signed out

The output of the weighted summer

wsum.*N***.bit.***M***.weight** signed rw

The weight of the *m*'th input of weighted summer *n*. The default value is 2ˆ*m*.

wsum.*N***.offset** signed rw

The offset is added to the weights corresponding to all TRUE inputs to give the final sum.

NAME

xor2 − Two-input XOR (exclusive OR) gate

SYNOPSIS

loadrt xor2 [count=*N***|names=***name1***[,***name2***...]]**

FUNCTIONS

xor2.*N*

PINS

xor2.*N***.in0** bit in
xor2.*N***.in1** bit in
xor2.*N***.out** bit out

out is computed from the value of **in0** and **in1** according to the following rule:

in0=TRUE in1=FALSE
in0=FALSE in1=TRUE
out=TRUE

Otherwise,
out=FALSE

LICENSE

GPL